APPLE II 使用手冊

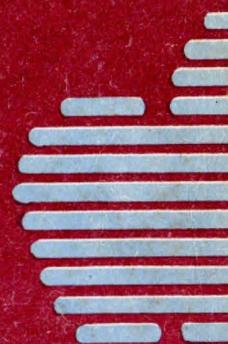
張其邦 譯













APPLE II
使用手冊

張其邦　譯



科藝出版社

APPLE Ⅱ使用手册

譯　者：張　　其　　邦

出版者：科　藝　出　版　社

發行者：科　藝　出　版　社

地　址：九龍西洋菜街102號三樓

印刷者：達　華　印　刷　廠

地　址：香港柴灣工廠大廈10樓

簡　介

這是一本爲您寫的有關Apple Ⅱ 的索引書，在本書中將要描述Apple Ⅱ 本身以及其它一些常見的週邊設備，還有像磁碟機和列表機等附屬用品。

這本書假設您可以使用一部Apple Ⅱ 系統，依您的手册所說地接上所有的系統配件；我們不會講解怎麼樣來裝配您的系統，而是專注於當您裝好了以後如何去使用它。

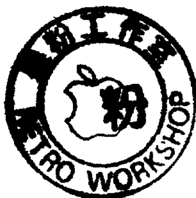
Apple Ⅱ 是個什麼玩意兒？您怎麼樣叫它幹活？本書的頭兩章爲您做個解答。也許您已經注意到Apple Ⅱ 系統是由好些個元件，利用電路板、電線、電纜等串在一起的，第一章的內容就是告訴您這些個元件各是些什麼，拿來做什麼用的。第二章說明如何去運用每一個元件，當有了這個知識之後，您就可以使用很多現成的，像是文件處理（WORD PROCESSING ）、財務分析、庫存、電腦教學、以及企業管理等等方面的程式了。

接下來的四章教您如何在Apple Ⅱ 上頭製作BASIC程式。第三章是一個基礎部份，我們討論如何使用Apple Ⅱ 的兩個BASIC版本——整數BASIC與 APPLESOFT 。第四章就包含了進一步有關BASIC的功能與程式技巧。另外，有兩項比較深一點的論題，即磁碟與螢幕圖像，應該有它們自己獨立的一章；第五章中解釋如何用磁碟來儲存您的程式與資料檔，第六章講的就是使用Apple Ⅱ 各個繪圖方式在螢幕上顯示出圖型的技巧。

BASIC 程式是由Apple Ⅱ 中的一個監督程式管理之下執行的，第七章所討論的就是標準監督程式與自動啓動監督程式，從一個BA-

SIC 程式員的觀點之下的一些問題與技巧，同一章中還告訴您如何在您的BASIC程式中與組合語言程式連用。

第八章是一個有關兩個BASIC版本中每一道敍述、函數、以及磁碟敍述的詳細說明，再加上各個附錄，本書爲您提供了一個對於如何在Apple Ⅱ上寫作BASIC程式的參考。



目　　錄

第一章　APPLE Ⅱ 的基本面目

第二章　如何操作APPLE Ⅱ



第三章　如何用BASIC語言寫程式

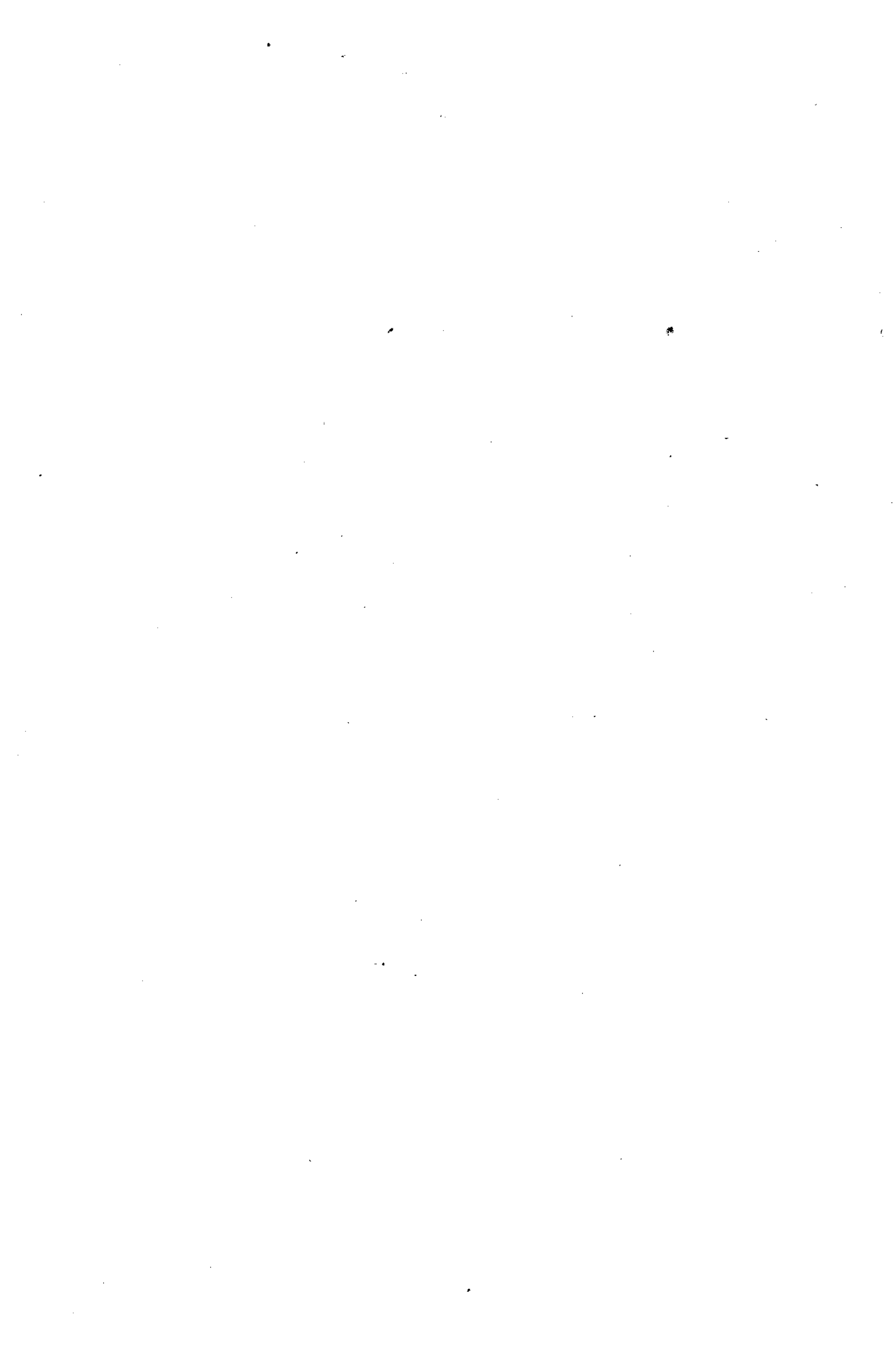
第四章　深入地瞭解BASIC程式語言

第五章 DISK Ⅱ 磁碟機

第六章 圖形與聲音

第七章　機器語言的監督系統

第八章　BASIC敍述與函數概要



1

APPLE Ⅱ 的基本面目

圖1－1是一個標準的APPLE Ⅱ微電腦基本組合。需要注意的是，這個完整的系統由很多分離的設備結合而成；由於有許多可以增加的週邊設備，而每個人的需要並不相同，使得個人擁有的系統也許並不如圖1－1所示。但是每個系統一定有以下三項設備：APPLE Ⅱ主機，與主機連接的鍵盤以及顯示螢光幕。現在讓我們對這些必需的設備以及一些可以供選擇的週邊裝置做進一步的描述。本書並不討論應該如何將這些週邊裝置與APPLE Ⅱ連接。要知道如何連接這些設備，使用者必須參考每項週邊裝置所提供的參考手册。

鍵盤及顯示螢光幕

我們可以利用鍵盤及電視螢光幕與APPLE Ⅱ做各種的聯繫。與打字機型式相似的鍵盤和APPLE Ⅱ主機連接在一起，經過它，我們可以將指令打入APPLE Ⅱ內。

顯示螢光幕可以是一般的彩色電視機或專爲電腦使用的顯示螢幕（

MONITOR）。黑白電視機當然也可以做爲顯示器，但彩色的圖形就無法照原色表現了。螢光幕不但顯示由鍵盤輸入的資料，而且忠實的表達出**APPLE Ⅱ**對指令的反映。

標準的螢幕有三種不同型式的顯示，主要的一種是以黑白色顯示文字資料，其他兩種則用於表現各種不同的圖形。在顯示字幕時，螢幕被分爲24行，每行40個字。顯示圖形時，則表現的是點與綫而非字形，螢幕被分得更細更密（關於圖形的顯示將在第七章做更詳盡的說明）。

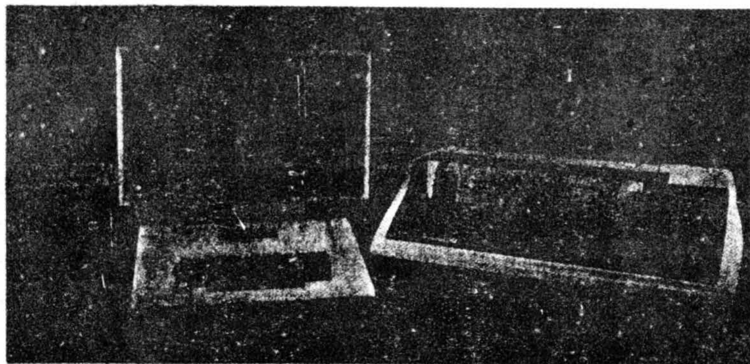


圖1－1　典型的APPLE Ⅱ電腦系統

大多數的**APPLE Ⅱ**擁有者都使用**MONITOR** 做爲顯示器。**MONITOR** 較電視螢光幕顯示的影像清楚得多，只是不能用它來看電視節目罷了。圖1－2是一個電視機與**APPLE Ⅱ**相連接的典型例子。

如圖所示，電視機並非與**APPLE Ⅱ**做直接的連接。一個可撥動的開關連接在電視機的天綫上，當開關被定在某個位置時，電視機呈現它正常的功能——接收電視台訊號，播映電視節目；而當開關被定在另一個位置時，則電視機就接受**APPLE Ⅱ**的控制了。一條電綫從開關連接到**APPLE Ⅱ**內部的一塊電路板，提供了控制訊號的通路——這就是一個**RF**轉換器（**RF MODULATOR**）、它的功能就是將

APPLE Ⅱ所傳送的視訊轉換成電視機所能接受的訊號。**APPLE** Ⅱ的經銷商將會銷售**RF**轉換器，並示範如何將電視機與**APPLE** Ⅱ連接。

專用顯示螢光幕則不需要**RF**轉換器就可以直接和**APPLE** Ⅱ連接。如同圖1－3所顯示的情形。

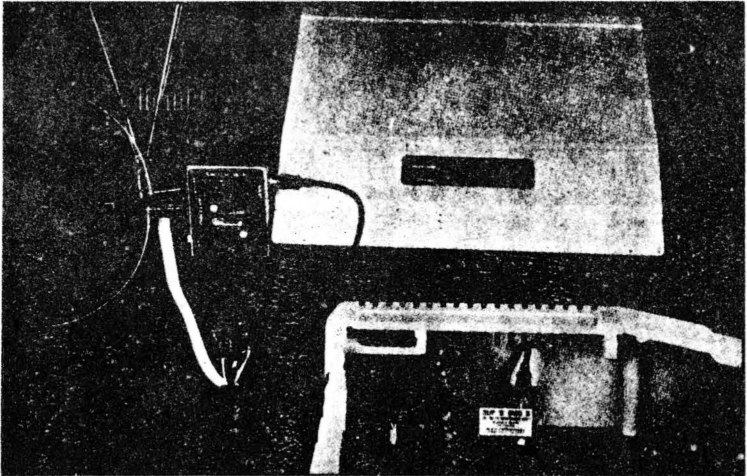


圖1－2　與電視的連接情形

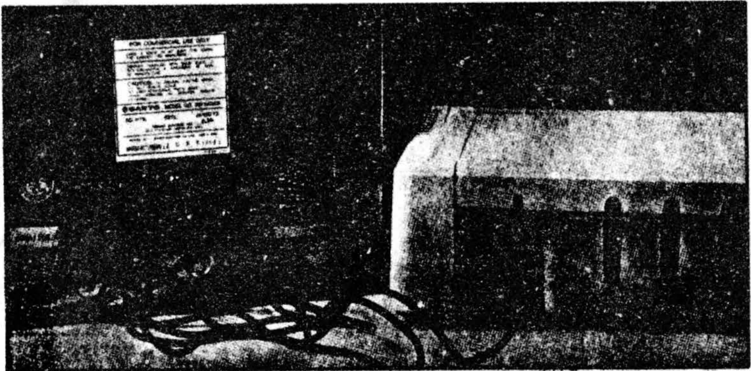


圖1－3　與顯示幕的連接情形

APPLE Ⅱ的內部

APPLE Ⅱ本身在使用者的控制下管理所有的週邊裝置。隱藏在鍵盤後面的是APPLE Ⅱ的記憶板、微處理機、各種週邊設備的連接點以及其他許多的電子設備。圖1－4顯示揭開封蓋板後的APPLE Ⅱ內部的情形。

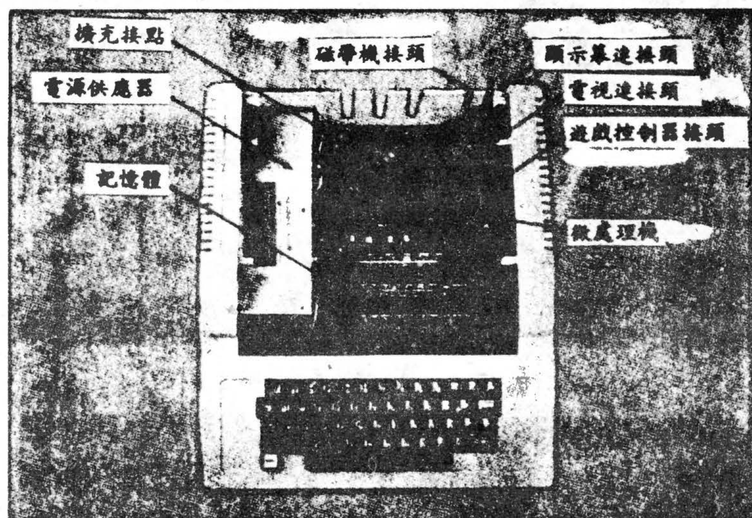


圖1－4　APPLE Ⅱ的內部

APPLE Ⅱ的內部也許因為每台機器所擁有的週邊裝置數目的不同而與圖1－4並不相合，但是基本的構造必定相同。大塊的線路板上佈滿了整齊排列的積體電路，還有一些較小的線路板垂直地插在主板後方的擴充接點(SLOTS)上。積體電路及垂直線路板的數目，因每台

機器而有所不同。

記憶體

電腦的**記憶體**（MEMORY）是由許多的小單位——**數元組**（BY-TE）組合而成的。記憶體的每一個**數元組**可儲存一個字母或相類似的資料。由於**晶片**（CHIPS）數目的不同，APPLE Ⅱ記憶體的大小可由4,096個數元組增加到65,536個數元組，也就是可以由4K數元組增加到64K數元組（1K數元組表示1,024個數元組)。記憶體的大小往往決定APPLE Ⅱ所能從事的工作及工作的性質，這點將在後節詳述。

APPLE Ⅱ本身具有兩種形式的記憶體。其一爲ROM（READ ONLY MEMORY）——僅讀記憶體；它的內容即使將機器關掉也不會改變。ROM保存有APPLE Ⅱ 的系統程式，具有辨認指令及做正確反應的功能。其二爲RAM（RANDOM ACCESS MEMOR-Y）——隨機處理記憶體；它的內容隨時可以被改變。實際上，一個存在RAM內的程式也就是APPLE Ⅱ可以隨時執行的程式。

需要注意的是，RAM只在開機後才能使用，當關機後所有的記憶將會消失無蹤。

磁帶機

磁帶機（CASSETTE TAPE RECORDER）可被用來儲存程式及資料，並可與RAM作各種資料的交換、聯繫；因此我們可以利用磁帶儲存許多的資料及程式。圖1－5顯示一般的磁帶機與APPLE Ⅱ連接的情形。

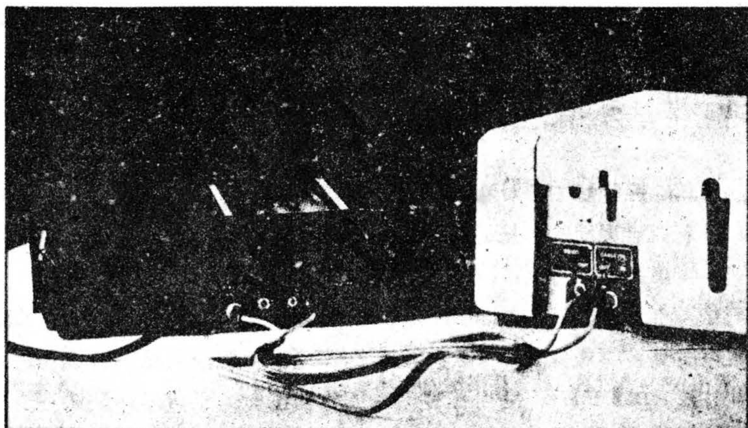


圖1－5　磁帶機的連接情形

磁碟機

以磁碟機（DISK DRIVE）做爲資料儲存器遠較使用磁帶機來得好用，因爲磁碟機處理資料的速度更快、可靠度更高而且儲存資料的量較磁帶機大得多。磁碟機有各種不同的型式及大小，每一種型式都有不同的儲存容量。圖1－6顯示兩個DISK Ⅱ的磁碟機（由APPLE電腦公司所生產）。

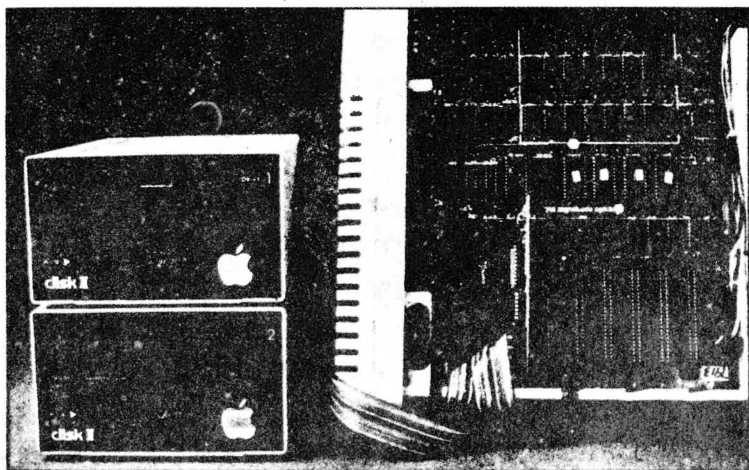


圖1－6 磁碟機DISK Ⅱ的連接

APPLE Ⅱ的軟體系統

現在讓我們從APPLE Ⅱ的硬體介紹轉向APPLE Ⅱ的軟體。目前並不討論使用APPLE Ⅱ所能從事的工作，例如：文字處理（WP），會計、財務分析以及其他的應用；現在所要討論的是不同型式的程式，由於這些程式的存在才能使APPLE Ⅱ處理以上的各種應用工作。一個能夠玩遊戲、寫信或做其他實際上應用的程式就被稱為應用程式（APPLICATION PROGRAMS）。應用程式可以由磁帶機或磁碟機傳輸到RAM內執行，儲存或修改，而非在ROM內。比方說，想要使APPLE Ⅱ成為一台專做文字處理的電腦，即可從磁碟機中將此應用程式傳送到RAM內而執行這項功能。我們將在第二章解釋如

何做這項傳輸工作。

程式設計師經常使用不同的程式語言（PROGRAMMING LA-NGUAGE）設計應用程式，程式語言對程式設計師來說似乎是簡單而好用的，但對APPLE Ⅱ而言則必須加一些幫助才能使它瞭解程式的命令。因此，一個特殊的程式——編譯程式就產生了，它將撰寫應用程式的程式語言翻譯成機器能夠瞭解的語言——機器語言。APPLE Ⅱ具有數種不同的編譯程式，可以被儲存在RAM內或是ROM內。

編譯程式必須與另一程式緊密配合，才能夠使APPLE Ⅱ的設備接受指揮；這程式一般被稱爲作業系統程式（OPERATING SY-STEM PROGRAM），它的功能在於執行一些基本的系統動作；例如：自磁碟或磁帶中傳輸程式及資料到RAM內，以及將鍵盤輸入的資訊忠實的顯示在螢光幕上。APPLE Ⅱ的作業系統程式被稱爲監督程式（MONITOR）。監督程式永遠只被儲存在ROM內。

週邊設備控制卡

需要注意的是插在APPLE Ⅱ主板後方擴充接點上的垂直線路板，擴充接點即爲了能使這些線路板與主機連接而存在的。這些線路板就稱爲控制卡（CONTROL CARD），控制卡上有許多的電子零件，使得APPLE Ⅱ能夠控制週邊裝置的動作；例如：磁碟機的控制卡提供了APPLE Ⅱ與磁碟機控制訊號的道路。

APPLE電腦公司製造了許多的控制卡都是插在主板後方的擴充接點上；有一些控制卡由於插在擴充接點上的位置相同，因此不能同時在一台機器上使用，但大部份的控制卡都可以被插在任何一個擴充接點上。每張控制卡都會將它的名字標示，但標示的位置往往在控制卡被插入擴充接點後就被遮蓋住了，而控制卡其他的地方都沒有特別記號，出廠日期或其他足資分辨的特別外觀，故而要分辨插在擴充接點上的是那一張

卡，就必需由它所插的位置及所接的週邊裝置是那一個來分辨了。如圖1－6所示。**DISK Ⅱ**的第一片控制卡必須被插在第六個擴充接點上，可以接一個或二個磁碟機；第三及第四個磁碟機可被接在第二片控制卡上，控制卡的位置是在第五個擴充接點上；至於第三片控制卡則可接第五及第六個磁碟機，它的位置則是在第四個擴充接點上。

現在讓我們看看**APPLE Ⅱ**的其他控制卡。

圖1－7所示的三張控制卡增加了**APPLE Ⅱ**的程式語言能力。**APPLE Ⅱ**依型式的不同而分別擁有整數**BASIC(INTEGER BASIC)**程式語言或實數**BASIC(REAL BASIC——APPLE－SOFT)**程式語言。但若在第0個擴充接點上加上適當的控制卡，則可同時擁有這兩種程式語言。語言系統卡(**LANGUAGE SYSTEM CARD**)及**APPLESOFT**語言卡(**APPLESOFT FIRMWARE**)可在任何機型上發生作用，但整數語言卡(**INTEGER BASIC CARD**)則是專爲**APPLE Ⅱ PLUS**所設計的。加上語言系統卡後，可以增加更多的程式語言系統，例如**PASCAL**程式語言即是一個例子。

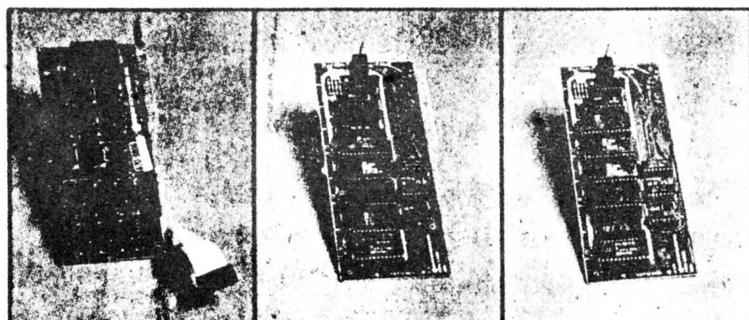


圖1－7　由左至右：語言系統卡、Applesoft語言卡及整數語言卡

圖1－8所顯示的是串列介面卡（SERIAL INTERFACE CARD），經常被放置在第一個擴充接點上，它最常用的功能往往是與列表機連接一起。

圖1－9顯示的是平行輸出列表機介面卡（PARALLEL PRINTER INTERFACE CARD），和上述的串列介面卡幾乎不可能共同使用，因爲它所放置的位置通常在第一個擴充接點上，而它的功能就是與列表機相連接。

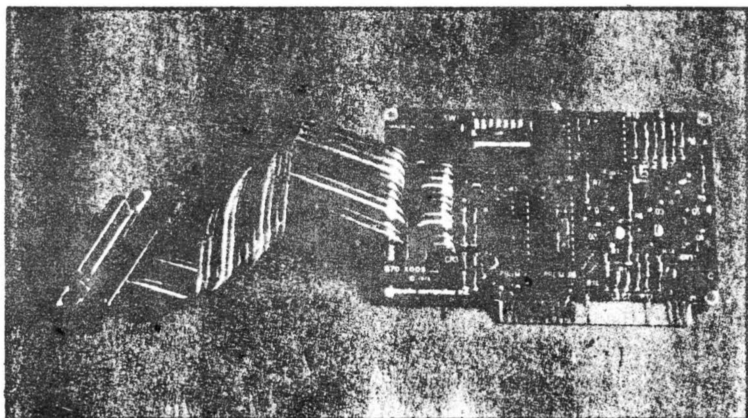


圖1－8　串列介面卡

圖1—10所顯示的是訊號交換介面卡（COMMUNICATIONS INTERFACE CARD），利用一個訊號轉換器（MODEM），將訊號交換介面卡與電話綫連接，使APPLE Ⅱ可以與遠方的電腦作資料交換的工作。

其他尚有許多的介面卡由不同的公司發展成功，提供了許多的功能。例如：控制燈光及電器設備的介面卡，以及可以用程式控制而產生樂器音響的介面卡；更有能將擴充接點數目增加的介面卡，它只佔

圖1－9 平行輸出列表機介面卡

圖1－10 訊號交換介面卡

APPLE Ⅱ擴充接點的一個位置，但却提供了許多的擴充接點供其他的介面卡使用。

有些**APPLE Ⅱ**由於加了一片特殊的介面卡，使得原先螢光幕顯示每行字的數目增加到每行80字（原本螢光幕顯示爲每行40字）；使用這介面卡時，顯示螢光幕必需與介面卡連接，而非與主綫路板連接。這個特殊的介面卡通常被插在第三或第四個擴充接點上。圖 1—11 顯示典型的這種介面卡。

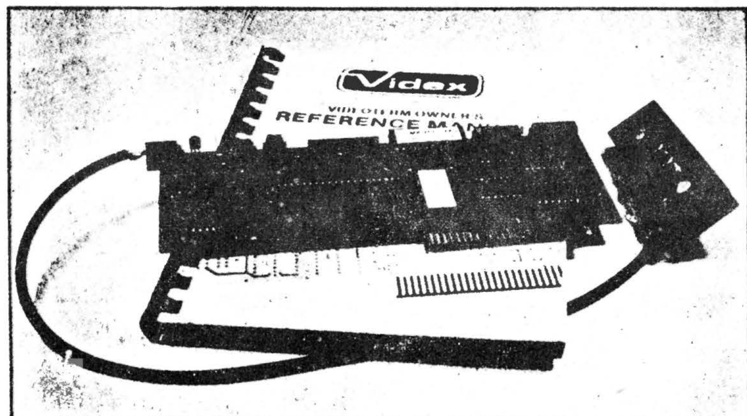


圖1—11 特殊顯示介面卡

遊戲控制器

我們最後所要討論**APPLE Ⅱ**內部設備的就是遊戲控制器（**GA-ME CONTROLLER**），圖1—12顯示它與**APPLE Ⅱ**連接的情形。遊戲控制器一般被使用在遊戲的控制方面，通常它並不與

APPLE Ⅱ連接，只是提供一個選擇的機會。

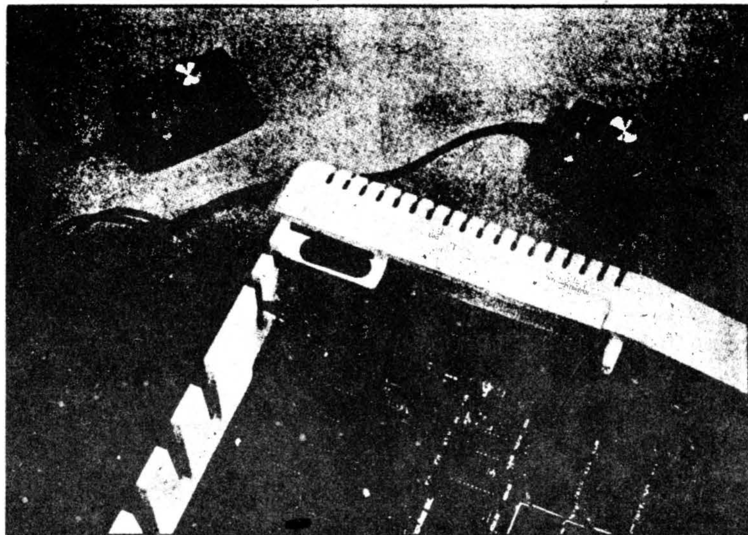


圖1－12 遊戲控制器的連接

列表機

現在讓我們介紹一下列表機(PRINTER)－如圖1－13所示就是一個列表機與APPLE Ⅱ連接的情形。

列表機由於種類的不同，可以分別利用串列介面卡或平行輸出列表機介面卡與APPLE Ⅱ連接；列表機具有許多不同的型式，分別具有不同的價格及不同的輸出功能。有些能夠印出如同打字機字型的字體，有些能夠將圖形列印在報表紙上(有些列表機甚至能夠印出彩色的圖形

）。當然，也有一些列表機兼具以上的功能。

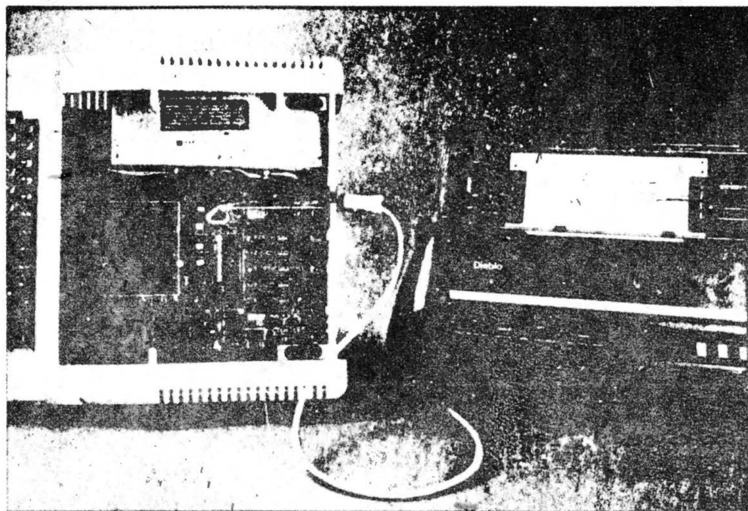


圖 1－13　列表機的連接

彩色繪圖板

彩色繪圖板（GRAPHICS TABLET）是由APPLE 公司發展成功的（如圖 1—14 所示）。它將APPLE Ⅱ的繪圖能力用特殊的方法表現出來——可以用手直接的操作。用它的〝筆〞您可隨意在繪圖板上劃出各式各樣的彩色圖形而顯示在彩色顯示螢光幕上。也可以利用它的固定功能，按照您的指示而劃出點、綫或實體的方盒；圖形可以佔有整個的螢光幕或只是某一部份，也可以被移動至您所希望的任何一個位置；圖形可以被放大、縮小或作分色處理，而在處理過程中，也可以

隨時將圖形儲存在磁碟片上，供以後的使用。至於計算圖形的面積或作距離的測量，也可以利用繪圖板的特殊功能達成這一個目的。

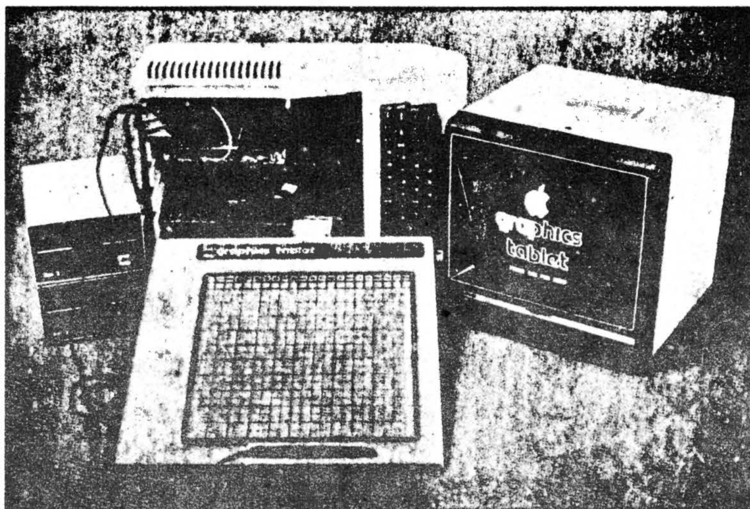
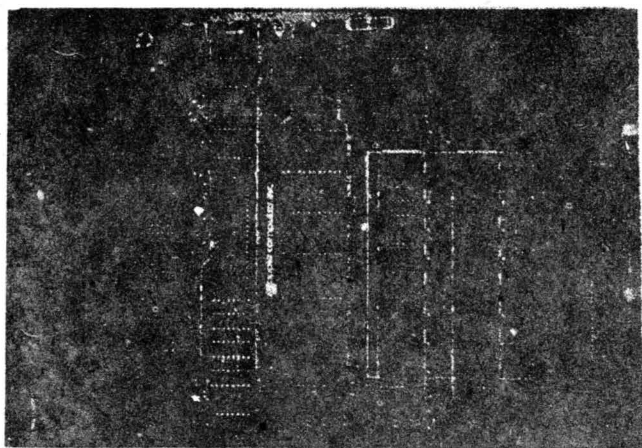
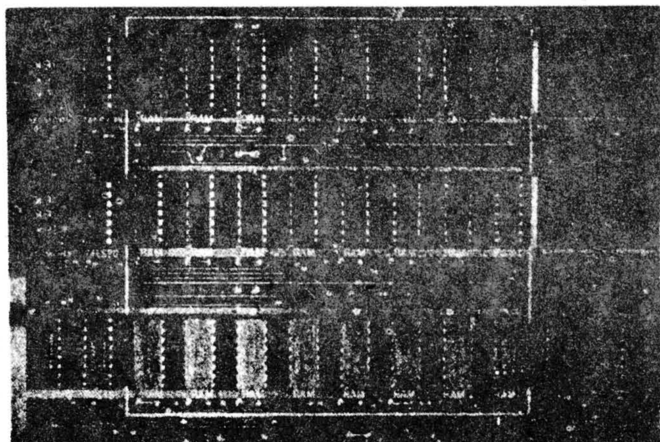


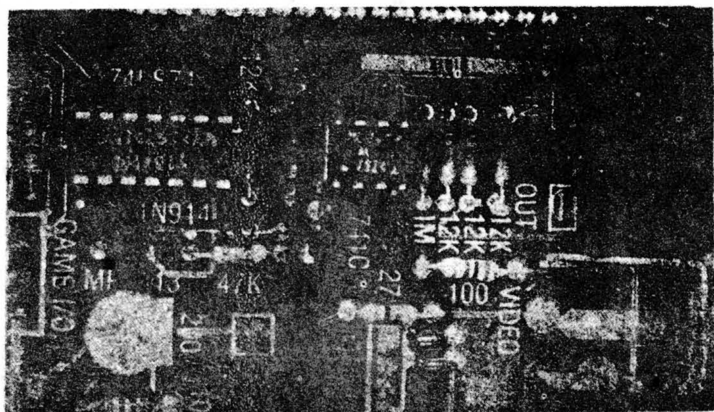
圖1－14 彩色繪圖板的連接



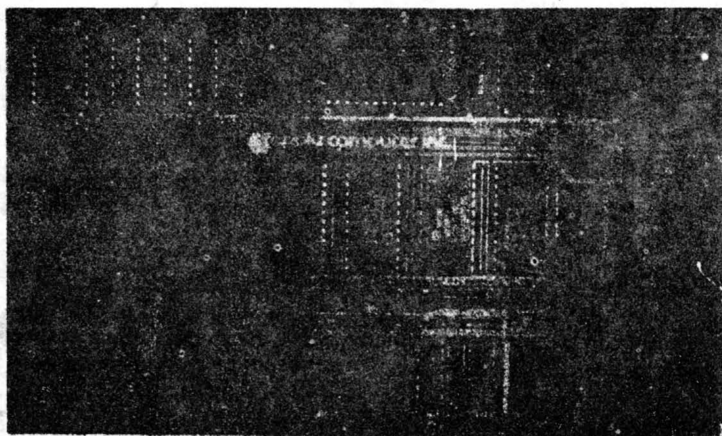
Apple II　母板俯視圖；右方爲鍵盤所在。



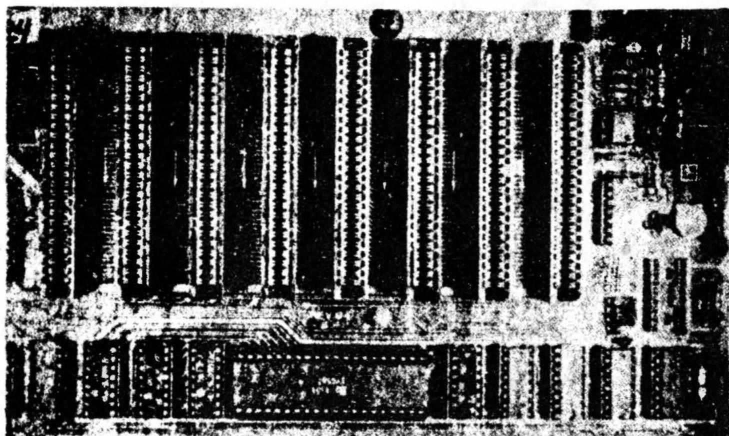
母板右下方白色框的放大。這是48K的記憶體所在地。



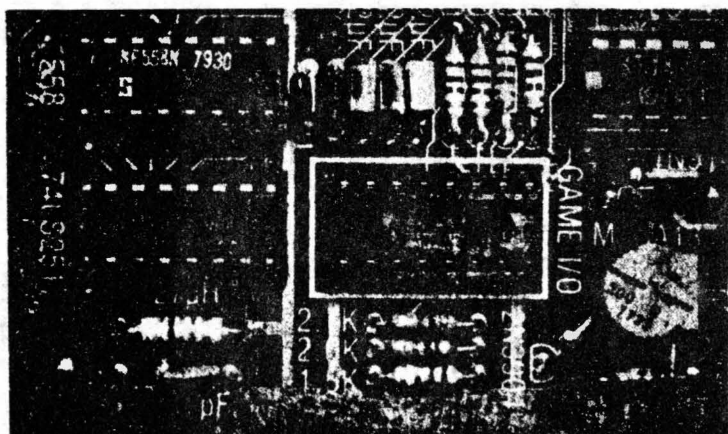
上圖是母板上的左上角部份，經順時針轉 180 度再放大而得，這是螢幕輸出的控制部份，右下角的插座就是用來與螢幕相連接的。



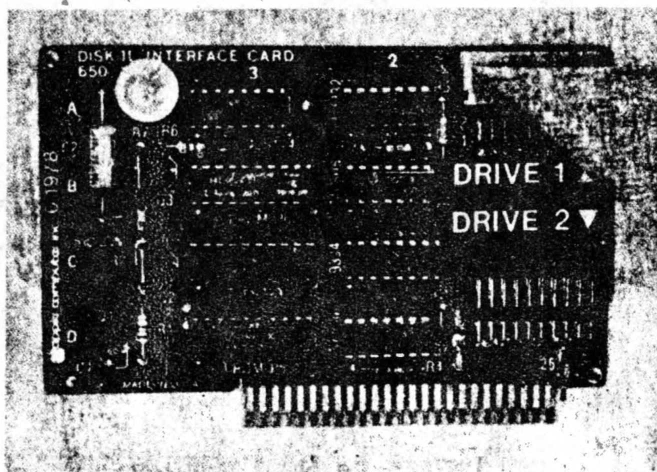
Apple Ⅱ 母板上 ROM 的部份(中排)，上排中央是6502 CPU，下排是 RAM 的一小部份。



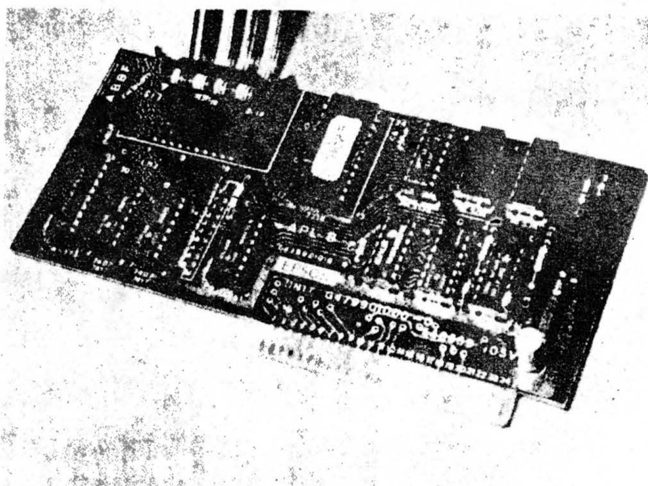
母板左方的八個接點，這兒最左邊就對應著母板上最下方一個，所有輸出輸入設備都經由這兒。自左而右，七個接點的編號是 0 到 7 ；下方的 IC 就是6502微處理機



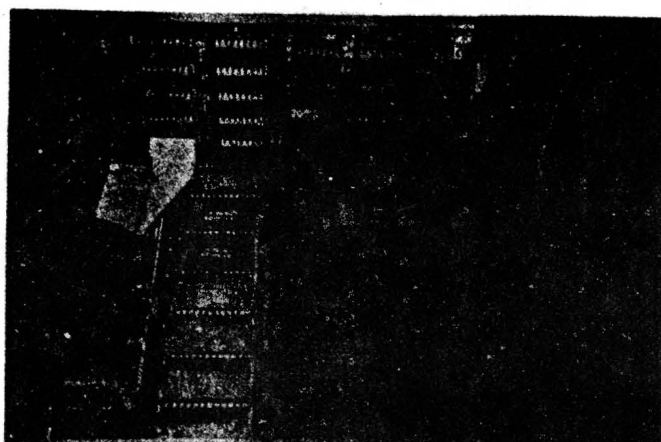
母板上左上角遊戲控制輸出輸入的放大圖。



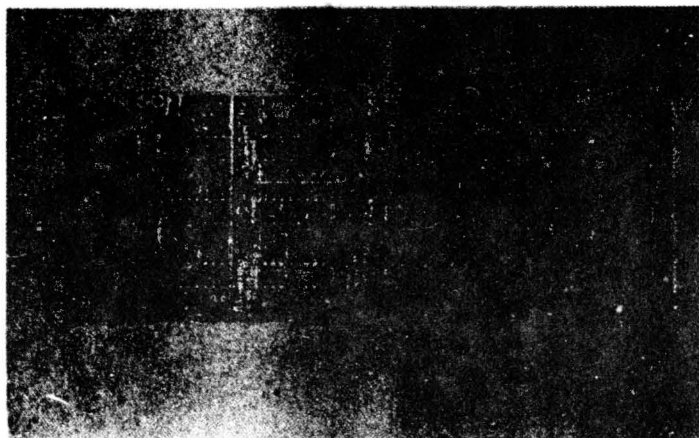
Apple Ⅱ 的磁碟機界面卡。



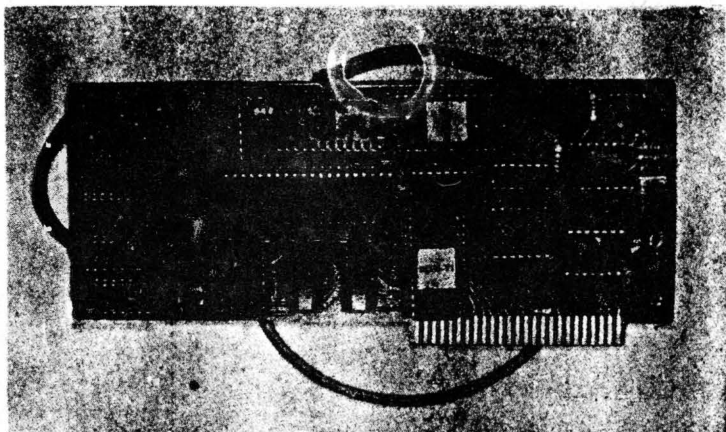
EPSON 牌列表機界面卡



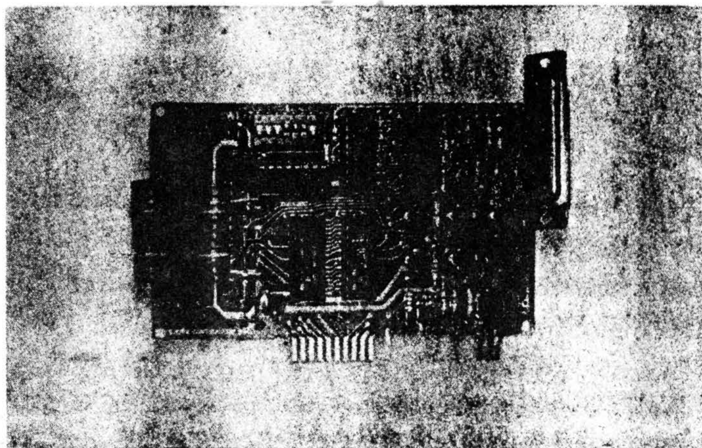
插在第0號接點上的語言卡，插上去了後就有了64K的記憶體，Applesoft與整數BASIC都會存在這兒。當有了它，您就可以使用Apple PASCAL 與Apple的FORTRAN-77了。



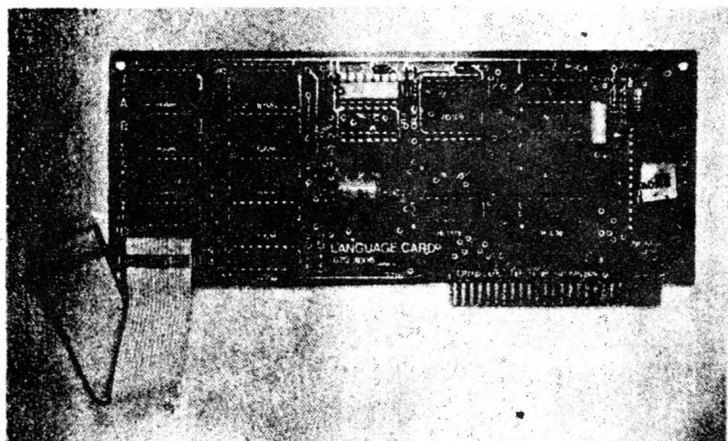
Microsoft公司爲Apple Ⅱ 製造的Softcard，上面有一個Z-80A的CPU；插在1到7其中一個接點上就會把Apple Ⅱ 變成一部Z-80的機器。隨這張卡一起附送的還有一套CP/M作業系統，一個MBASIC 編譯程式，這樣讓Apple Ⅱ 用戶加入CP/M園地，如虎添翼。



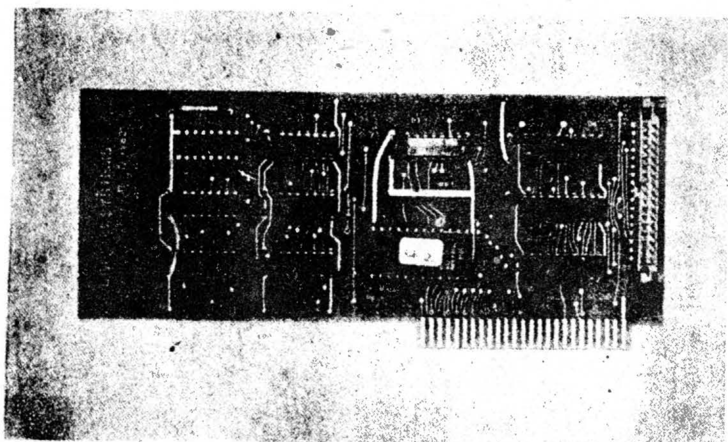
Apple Ⅱ 的螢幕有25列40行，加上這塊所謂的"80行界面卡"之後，螢幕上就變成有25列80行，並且可以顯示小寫字母。



這是Apple Ⅱ 的RS-232串列輸出界面卡。

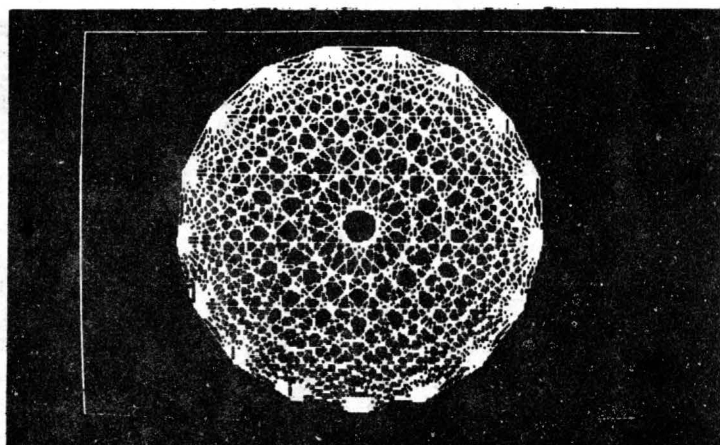


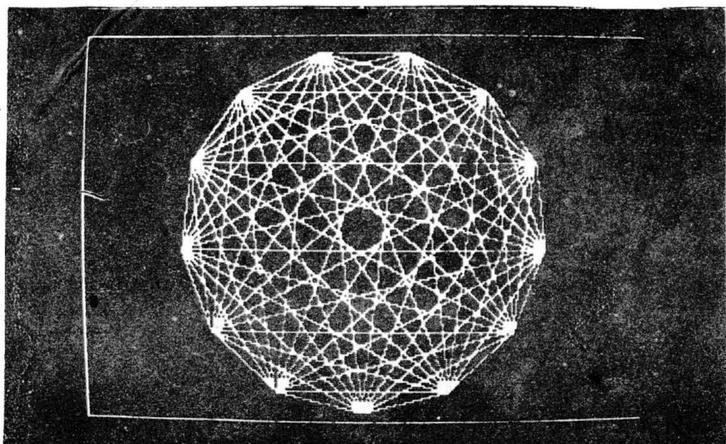
這是片 Apple Ⅱ 的語言卡，前面已經講過了，它得插在第0號接點。

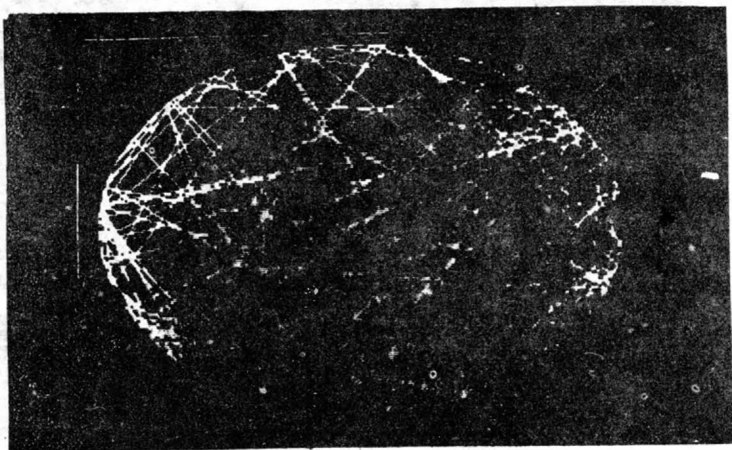


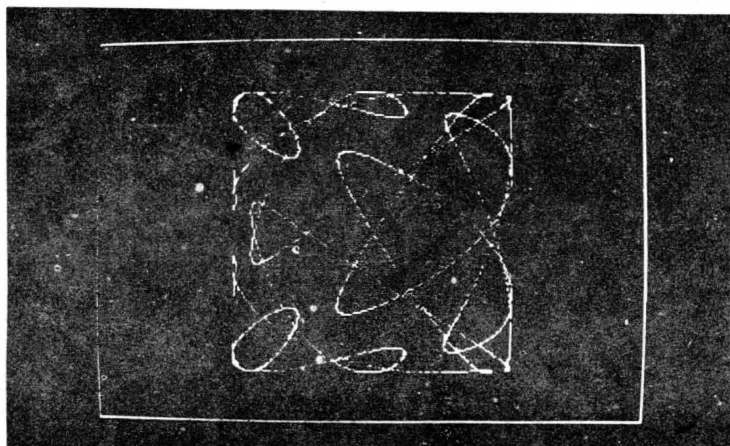
這片是 CORVUS 硬磁碟系統卡。這一型的硬磁碟有五百萬、一千萬、兩千萬等三個容量，它能夠配合 Applesoft，以及 Apple PASCAL Apple FORTRAN 使用；另外，經過〝多工器〞，還可以接成一套最多64部 Apple Ⅱ 的星狀網路。

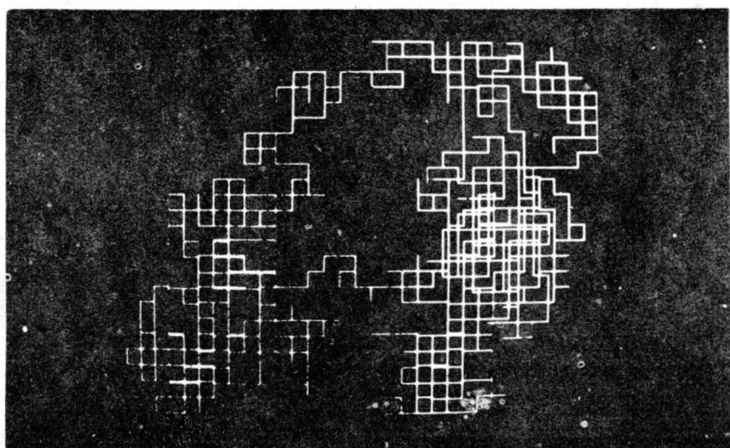
以下是在 Apple Ⅱ 上以高解析度功能，删掉最下方四列文字幕以後，所顯示出來的圖型。











2



如何操作 APPLE Ⅱ

當您第一次坐在一個電腦系統前面時，往往會有一些畏懼的心理，縱使是已經連接完成的完整APPLE Ⅱ系統，也免不了會有這種感覺。我們在這一章將會就如何使用APPLE Ⅱ做一番解釋，相信必能使您覺得輕鬆些。至於如何將系統建立完成，我們並不作說明，您在各種設備的使用手册中可以學習到建立系統的程序及方法。如果您懷疑您所作的程序是否正確，可以就近請教擁有與您相同設備的人士或是APP－LE Ⅱ的經銷商。

打開 APPLE Ⅱ 的開關

檢查系統的設備是否都正確地連接起來，然後打開電視螢光幕的開關，並將聲音關掉；然後將連接在天綫上的RF 轉換器的開關定在GA－ME或是COMPUTER的位置。再按照RF 轉換器的說明，將電視機的頻道定在指定的電視頻道上。如果您不知那一個頻道才是正確的，您可以請教 RF 轉換器的經銷商。

APPLE Ⅱ的開關就在APPLE Ⅱ的後方接近電源綫的附近；當您將開關打開後，立刻可以聽到"嗶"的一聲，這聲音就是告訴您APPLE Ⅱ已經準備好了，在鍵盤左側的POWER 燈除非燒壞否則也會亮著。

如果您沒有聽到"嗶"的聲音，請將開關關掉，然後再重新打開，如果您仍然沒有聽到任何聲音，將開關關掉，並看POWER 的燈是否會閃一下，如果不會，則將APPLE Ⅱ電源綫拔掉，檢查插座是否通電，如果都沒問題，您必須重新按照指示檢查所有的設備是否連接正確，接頭處是否接得緊密，然後再重新開機，如果仍然不能使APPLE Ⅱ有所動作，您必須立刻將開關關掉，並請APPLE Ⅱ的經銷商協助，千萬不要自己亂動機器的內部，因為這麼做往往會造成機器本身很大的傷害。

TV上的顯示

當您將開關打開並聽到"嗶"的聲音，這就表示一切都是正常的，這時在螢光幕上將會顯示出正常的圖形。

由於APPLE Ⅱ型式的不同，顯示在螢光幕上的圖形也就不同，但在螢幕上一閃一閃的一個小方塊却是相同的，這閃動的小方塊就叫做游標（CURSOR）。它所在的位置就是您所要打入的下一個字的位置，下面的一節將解釋如果您沒有看到游標時應該如何做。

如果您沒有看到游標

如果您的APPLE Ⅱ系統中加上了語言系統卡並與一個或多個磁碟機相連接，則在開機後，將不會看到游標，而會聽到轉動磁碟機及磨擦的吵雜的聲音，而且在磁碟機前板上的IN USE的燈將會亮著。在這種

情況下，您必需按 RESET 鍵，這時游標將會重現而磁碟機將會停止轉動，於是您就可以繼續下一節了。

如果您在APPLE Ⅱ內加上了能使螢幕顯示每行80個字的特殊卡片，您必須按照以下的指示打進一些命令，才能在螢幕上看到游標。

(一)按住CTRL鍵，壓緊不放，然後按B鍵，再同時放開。

(二)執行下一個步驟之前，您必須知道這張特殊卡片插在那一個擴充接點(SLOT)上，一般都是在第三或第四個擴充接點；您可以打開封蓋板看個確實——需要注意的是擴充接點的數目是由左至右分別由ø到7。圖1—11告訴您所要尋找的是那一片卡片。

(三)如果您的顯示螢光幕與位在第三個擴充接點上的控制卡相接，則您必需打入PR#3，然後按RETURN鍵；如果這控制卡是插在第四個擴充接點上，則打入PR#4，再按RETURN。

(四)這時候游標將會出現，雖然它可能並非一閃一閃的，但一切都沒有問題，這時您就可以繼續按照指示往下做了。

系統的標示

在游標的左方就是系統的標示字(PROMPT CHARACTER)，標示字位於每一行的第一個位置；它可能是"*"，">"或"]"中的任一符號。APPLE Ⅱ具有多種語言的功能，標示字就代表現在所能接受的程式語言。表2—1顯示這三種不同的標示字及它們所代表的程式語言。

"*"表示監督程式

當您使用APPLE Ⅱ時，也許在您一開機後，螢幕上所顯示的是"*"符號，如果您所用的機器，開機後並非如此，則您可以跳過這一

節。

表2－1 標示字

標　示　字	語　　言
*	監督程式
>	整數 BASIC
]	Applesoft

出現“＊”符號表示系統目前處於編輯監督程式的狀況，一般說來只有對APPLE Ⅱ有較精深研究的使用者，才會想要與APPLE Ⅱ的監督程式作直接的聯繫。如果您希望對監督程式做個練習，您可以任意爲之，但若您要重新使用正確的監督程式時，您必須關機然後再開機，使您所作的實驗，不致影響到系統的完整。

當“＊”符號出現時，您必須使APPLE Ⅱ回到BASIC語言系統內。

使用CTRL-B回到BASIC

我們可以使用一個監督程式所提供的命令使APPLE Ⅱ回到BA-SIC系統內；壓住 CTRL ，再按B鍵，然後同時放開，再按 RET-URN 鍵，您就會得到一個“＞”或是“］”的系統標示，表示您已進入BASIC語言系統內了。

BASIC是BEGINNER'S ALL-PURPOSE SYMBO-LIC INSTRUCTION CODE的縮寫，是一種廣泛被採用的電腦語言，它的發源地是在DARTMOUTH UNIVERSITY。APPLE Ⅱ依其型式不同分別具有整數BASIC或APPLESOFT 語言，這兩種型式的BASIC都較原先DARTMOUTH BASIC 具有更強的功

能。

“>”表示整數BASIC

當我們在螢光幕上看到“>”符號時，表示APPLE Ⅱ是處於整數BASIC系統下，APPLESOFT與整數BASIC的規則有許多不同的地方，但也有一些是相同的；就目前而言，我們所討論的指令都可以在這兩種語言上使用，所以您不必擔憂您所用的是那一種BASIC。

“]”表示APPLESOFT

“]”符號表示APPLE Ⅱ現在所能接受的程式語言是APPLE-SOFT；目前您並不需要擔憂您所使用的是那一種BASIC，在適當的時機，我們將會討論整數BASIC與APPLESOFT的區別。

APPLE Ⅱ的鍵盤

APPLE Ⅱ的鍵盤（如圖2－1所示）看上去與一般打字機的鍵盤相似，但較一般打字機多了五個鍵。在左邊有兩個分別是 Esc 鍵及 Ctrl 鍵，另外三個位於鍵盤的右邊，分別是 Reset 鍵，←鍵及→鍵。在右邊有另外的兩個鍵也許也不認識，它們分別是 Return 鍵及REPT鍵。

您可以儘管在鍵盤上打入任何東西而不會對APPLE Ⅱ產生任何損害，因爲任何不當的輸入都可以由一道手續恢復－－那就是將機器關掉再打開。

當您在鍵盤上打字時，您將發現您所輸入的字都是用大寫字體（不論您是否按住SHIFT鍵）顯示在螢光幕上，標準的APPLE Ⅱ僅僅顯示大寫字體，如果您想要顯示大小寫字體，您必須將您的顯示螢光幕

與能夠顯示大小寫字體的控制卡相連。當然也有一些程式知道如何去顯示大小寫字體。

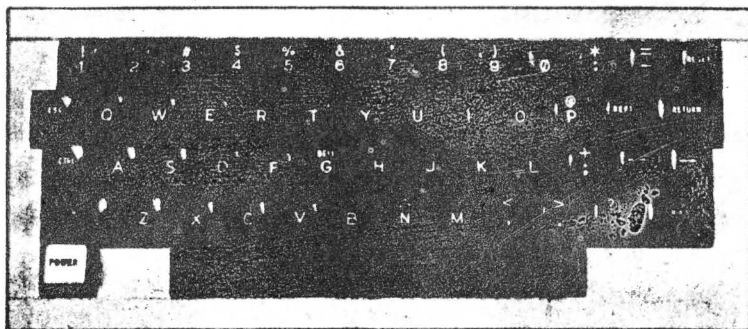


圖2－1 APPLE Ⅱ鍵盤

RESET鍵

在APPLE Ⅱ的鍵盤中，RESET 鍵是一個特殊的鍵；當您按了RESET 鍵，APPLE Ⅱ會停止任何正在處理的事，而系統的控制將會轉回到鍵盤。由於所使用APPLE Ⅱ型式的不同，按 RESET 鍵後分別會回到監督程式系統、整數BASIC系統或APPLESOFT系統。

有時按 RESET 鍵會產生很多的問題，特別是在使用磁碟機時，因此您必須練習避免不小心碰到RESET鍵，尤其須要注意的是在您想要按RETURN鍵時，因爲 RETURN 鍵與 RESET 鍵非常接近，一不小心就容易按到 RESET 鍵。有些型式的APPLE Ⅱ爲避免這種傷害，堅持必須與 CTRL 鍵同時使用（可參考下面 CTRL 鍵的討論）。

RETURN鍵

當您在鍵盤上打字時，所打入的字顯示在螢光幕上，同時APPLE

Ⅱ將您輸入的字存在記憶體內，但並不做編譯的工作，直到您按下 RE-TURN 鍵。按 RETURN 鍵使得APPLE Ⅱ知道您已打完了您所打的一行，APPLE Ⅱ會自動將游標以後所有的字消除，然後辨認您所打入的這一行字，如果它能認識您所打入的字（一個指令），它將會作正確的反應，否則您將會聽到"嗶"的一聲並看到以下的任一種訊息（或者是其他的錯誤訊息）：

```
?SYNTAX ERROR
*** SYNTAX ERR
```

這時，APPLE Ⅱ讓您知道您所輸入的字它並不認識，您必需重新輸入（沒有任何錯誤的輸入能夠瞞得過APPLE Ⅱ）。

SHIFT鍵

對標準的APPLE Ⅱ而言，它永遠是顯示大寫字體的，故而 SH-IFT 鍵並非在分辨大小寫字體，而是使同一個鍵可以輸出兩種不同的字；您可以按住 SHIFT 鍵再按一個鍵而得到一個字，也可以放掉 SHIFT 鍵，按同一個鍵而得到另一個字。按住 SHIFT 鍵而得到的字，就是在該鍵上方的字。

我們使用 SHIFT -表示同時按下 SHIFT 鍵與另一個鍵；例如：SHIFT -3 得到的就是"#"字。

有些字的鍵，雖然您按住SHIFT鍵再按該鍵，並不會顯示出來。例如：SHIFT -N 顯示"∧"，SHIFT -P 顯示"@"。至於G鍵，雖然它的上方是BELL 字，但當您使用 SHIFT -G 並不顯示BE-LL而仍然是G字。

CTRL鍵

CTRL 就是CONTROL的縮寫。CTRL 鍵與 SHIFT 鍵相同，永遠與其他的鍵一同使用。當您使用 CTRL 鍵時，按住 CTRL 鍵然後再按其他鍵，直到放鬆此鍵後，才離開 CTRL 鍵。在本書中我們使用 CTRL－ 表示某個鍵與 CTRL 鍵合用。例如：CTRL－B表示同時按住 CTRL 鍵及B鍵。

CTRL 鍵像 SHIET 鍵一樣，增加了一些鍵的功能。G鍵是唯一的一個在它的上方表示出它與 CTRL 鍵合用時所產生的功能的鍵，那就是BELL 。當您打入 CTRL－B時，您可清楚的聽到"嗶"的一聲；這個命令使得APPLE Ⅱ的發聲器發生作用。

還有許多的鍵與 CTRL 鍵合用，產生不同的功能；例如前面所提的 CTRL－B，使得APPLE Ⅱ由監督程式系統轉入BASIC語言系統。

另一個常用的組合就是 CTRL－X， CTRL－X告訴APPLE Ⅱ您剛才所打的那一行要作廢，您想要重新開始。您可以試著打一些東西然後使用 CTRL－X，您將發現游標將會跳回到那一行的開始處，您剛才所打入的字仍然存在，但APPLE Ⅱ並不理會它們是什麼，因爲就APPLE Ⅱ而言，您等於並沒有打入任何東西。

Esc鍵

Esc 鍵有許多的用途，有一些在本章中將會敍述，另一些則在下一章作解釋。

Esc 鍵與 CTRL 鍵及 SHIFT 鍵不同的地方在於：當您使用 Esc 鍵時不必一直壓住 Esc 鍵而是壓住後可以立刻放開，再按需要配對的鍵；這個程序稱爲ESCAPE序列。

最簡單的ESCAPE序列就是 Esc－@，您首先壓住然後離開 Esc，接著使用 SHIFT－P（SHIFT－P爲@），它的結果如何？您將看到在螢光幕上的一切都消失無蹤，僅留下游標在螢光幕的左上角

閃動。（在電腦的術語中，螢光幕的左上角就叫做母位（HOME）。

←鍵與→鍵

這兩個箭頭鍵叫做左箭號（LEFT-ARROW）與右箭號（RIG-HT-ARROW）。您將會發現←及→鍵是非常有用的，因爲它們准許您更正任何打錯的資料，或更正您已經打入的資料。←鍵如同打字機上的退位鍵（BACKSPACE KEY），您每按它一次，在游標下的字就不再存在記憶體中了，而游標就會倒退一格。您可以試著打入一些字（例如PRINT），然後按←鍵數次，直到游標回到開始打的地方，您可以看到在螢光幕上的字並沒有消失，但您可以確定的是，APPLE Ⅱ已將它們從記憶體中洗掉了。試著將游標移回到那一行開始的地方，然後再按←鍵，您將會發現游標跳到下一行，而且多了一個系統標示的符號。

如您所猜測的，→鍵使得游標沿著同一行往右邊移動，當游標往右移動時，它所經過的字就如同您重打過一般地存在記憶體內。您可以打入幾個字，然後使用←鍵回頭移幾位格子，再用→鍵使游標向右移動。當您每次使用→鍵時，游標所經過的字，就會被儲存在記憶體內。

REPT鍵

REPT 就是REPEAT的縮寫。如果您同時按住 REPT 鍵及任何一個鍵，這個鍵就會連續地重複顯現在螢幕上，直到您鬆掉其中一個鍵或是同時放掉這兩個鍵爲止。當與 REPT 鍵合用的鍵是←或→時，而您想要消除或重新加入的字很多時，您將會發現 REPT 鍵使您方便得多。爲了要使重複鍵（REPT 鍵）使用得正確，您必須先壓住所要重複的鍵，而當您需要停止時只要鬆掉 REPT 鍵就可以了。

其他的鍵

APPLE Ⅱ鍵盤上的其他鍵，相信您必定是非常熟悉的，因爲它們都是一些英文字，0～9的數字及一些標準的符號。

許多的打字員無法分辨數字的0（零）與字母的O或是數字的1與小寫字母的l。APPLE Ⅱ無法使用0代替O或是用1代表l，因此爲了使您便於記憶，APPLE Ⅱ的鍵盤在0上加了一條斜綫，同樣地，當0顯示在螢光幕上時也會有一條斜綫。

磁帶機

如果您的APPLE Ⅱ擁有磁帶機，則您可以從磁帶中將程式傳輸到APPLE Ⅱ內。有些錄有程式的磁帶會隨著APPLE Ⅱ到達您手中，您也可以買些其他的程式，當然如果您有興趣，您也可以自己製作（在第四章我們將告訴您如何做）。

如何處理磁帶

由於磁帶很容易受到損壞，受到損壞後又不容易恢復，因此您必須練習如何小心地去處理它。

需要注意的是，千萬不要碰到磁帶的表面，因爲不論您的皮膚多麼乾淨，皮膚上產生的天然油脂也會汚染磁帶。當磁帶不使用時，記住，一定要將它放回到盒子裏去，放置的地方不可以太熱，要遠離具有磁性的東西，更不可直接受到陽光的照射。

標示每一個磁帶

在每一個磁帶上您應該標示所錄上去的程式，這個工作避免您往後尋找時的許多困擾。

如何保護存在磁帶中的程式

每一個磁帶的底部都有兩個切痕（NOTCH），而大多數的磁帶機都能分辨出這兩個切痕，當它們存在時，磁帶機將不能再記錄任何東西到磁帶上。空白的磁帶往往有膠帶貼在這兩個切痕上，使得想要存入的東西能夠反覆地錄進去。因此如果您想要保護在某些磁帶上的重要程式，您只要撕掉切痕上的膠帶，使切痕暴露在外就可以了。

如何決定兩個切痕中的那一個是您想要保護的程式所在的那一面呢？您可以拿著磁帶，面對著磁帶的底部，將存有您所要保護程式所在的那一面向上，這時您只要撕掉右邊的那一片膠帶，使切痕暴露於外，您的程式就被保護住了。

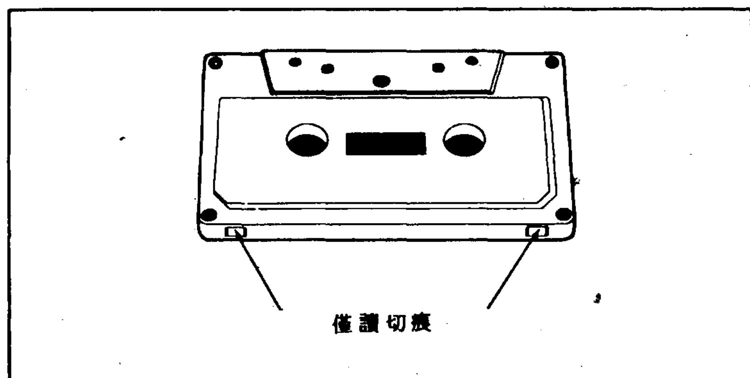


圖 2－2　磁帶保護切痕

如何調整磁帶機的音量

在您想要將程式傳輸到APPLE Ⅱ內時，首先必須先將磁帶機的音量調整到適當的程度，因爲如果音量太高或太低時，存在磁帶上的資料傳輸出來時會有變化而使APPLE Ⅱ不能瞭解。

目前唯一的一個方法來決定究竟多高或多低的音量對APPLE Ⅱ而言是適合的，只有依賴不斷的試驗、修正。現在告訴您一個常用的程序；首先，將音量調到最低，然後試著傳輸程式進入APPLE Ⅱ內，如果不能成功，則將音量稍稍提高，再試著做傳輸的工作，如果仍然不能成功，則繼續將音量提高，再做傳輸；如此反覆地試驗，直到成功地將程式傳輸入APPLE Ⅱ內爲止。

也許您購買的APPLE Ⅱ系統內包含有若干卷磁帶；如果您擁有的APPLE Ⅱ的系統語言是APPLESOFT ，則請您找出標示有" COLOR DEMOSOFT "的磁帶，如果是整數BASIC語言系統，則找出" COLOR GRAPHICS"的磁帶。將找出的磁帶放到磁帶機內（必須確定磁帶的標示朝上）。

現在您可依照以下的程序將程式傳輸入APPLE Ⅱ內。

㈠將磁帶轉回開始的所在。
㈡在APPLE Ⅱ的鍵盤上打入LOAD指令。
㈢在磁帶機上按下PLAY鍵，使磁帶開始轉動。
㈣按下RETURN鍵。

在您按下RETURN鍵後，游標會消失；大約15至20秒的時間內，您可以知道是否成功地完成了傳輸的工作。

如果您得到" ? SYNTAX ERROR "或" * * * SYNTAX ERR "的訊息，不要去調整音量，只是再按照以上的步驟重做一次，

如果這種錯誤的訊息繼續地發生，試著將磁帶機的讀寫頭清潔一下或換另一捲磁帶。

如果APPLE Ⅱ沒有任何反應或得到〝ERR〞或〝ERRERR〞的訊息，則您必須按RESET鍵並調整磁帶機的音量，然後再重新試一遍。

如果您聽到〝嗶〞的一聲而且沒有任何的訊息顯示在螢光幕上，這就表示一切都沒有問題，APPLE Ⅱ已經找到程式的起頭而且正將它傳輸到記憶體內。大約再過15秒或更長的一段時間（時間的長短依賴程式大小而定），將會再有〝嗶〞的聲音而系統標示及游標又再度出現，您可以停止磁帶機的轉動，並在音量的旋鈕上做個標示，省得您以後再使用時又得調整的麻煩。

當您看到系統標示及游標時，您可以知道程式已由磁帶上頭成功地傳輸到APPLE Ⅱ內了。

如何使用磁碟機

如果您的APPLE Ⅱ系統中包括了磁碟機，那麼您可以由磁碟片中傳輸程式，而不必使用磁帶了。APPLE Ⅱ的磁碟機提供了一片叫做〝SYSTEM MASTER DISKETTE〞的磁片，裏面除了APPLE公司在碟帶上所提供的程式之外，還包含了一些特別爲DISK Ⅱ磁碟機設計的程式。

如何處理磁碟片

由於磁碟片較磁帶更加脆弱，您在使用磁碟片時必須特別小心；絕對不要彎折磁碟片，也不要碰到磁碟片露在外面的地方，更不要用力擠壓磁碟片進入磁碟機。當您將磁片拿出磁碟機後，一定要將它放入封套

內，並記得遠離太陽的照射，及有磁性的東西，也不要將它放在很熱的地方，尤其對於" SYSTEM MASTER DISKETTE "特別要注意。

如何將磁碟片放入磁碟機內

圖 2 — 3 告訴您如何將磁碟片放入磁碟機內。您使用拇指及食指拿住磁碟片（拇指在上），打開磁碟機上的門，然後輕輕地將磁碟片滑入磁碟機內；一般來說，這個過程都應該順利完成而不會受到任何阻力，如果碰到任何阻力，您可以將磁碟片拿出來，再重新放入；需要注意的是，您在放入磁碟片時，要儘量的保持磁碟片的水平。當磁碟片進入磁碟機內後，您就可以將磁碟機的門輕輕地關上了，這個過程應該沒有問題，但如果無法輕易地將門關上，那您必須將磁碟片完全地推入磁碟機內，再試著關上門；如果您未能把磁碟片正確地放置在磁碟機內而用力地將門關上，那將會毀掉磁碟片。有時您可以等待磁碟機轉動後再關上門，因為磁碟機轉動後會使磁碟片放在磁碟機中央的位置了。

磁碟操作系統

在您使用磁碟機之前，有一個特殊的程式——磁碟作業系統（DISK OPERATING SYSTEM）必須先存入APPLE Ⅱ的記憶體內。磁碟作業系統（本書中簡稱為 DOS）控制了所有與磁碟機相關的動作；這種將 DOS 存入記憶體的動作，一般被稱為" BOOTING "，在電腦的術語中，您可稱這個動作為" BOOT THE DISK "或是" BOOT THE DOS "或" BOOT DOS "。

當您將機器關了再開時，您必須重新" BOOT DOS "。

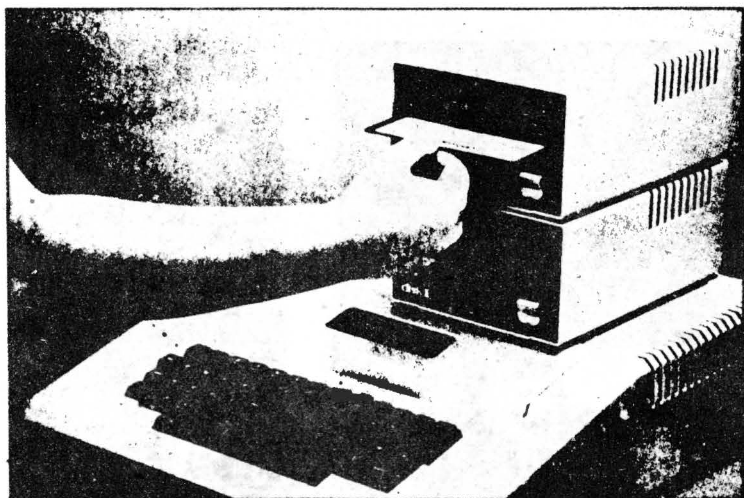


圖2－3　將磁片插入DISK Ⅱ內

如何將DOS存入記憶體內

依據您所擁有的APPLE Ⅱ系統的不同，有許多種不同的方法將DOS 存入記憶體內，每一個方法都假設磁碟機連接在第六個擴充接點上的控制卡的第一個連接點上，圖2－4顯示連接的情形。

將"SYSTEM MASTER DISKETTE"插入第一個磁碟機內，然後關上門。若您使用了語言系統卡，在某些情形下，您必須在使用DOS磁碟片前，先用"INTEGER AND APPLESOFT Ⅱ"磁碟片，這種情形將在下節詳述。

當您成功地將DOS存入記憶體時，螢光幕應該如同圖2－5中顯示的任何一個情形。

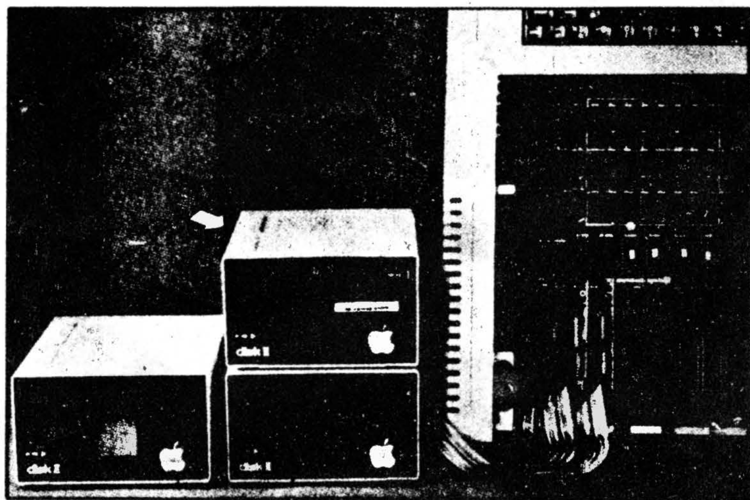


圖 2－4 標準的BOOT磁碟機

自動將DOS存入記憶體內

最簡易的方法就是自動將 DOS 存入記憶體內（AUTOSTART BOOTING），如同名字所顯示的，一切都是自動完成的。如果您的系統監督程式是自動的（AUTOSTART MONITOR），那麼您的機器就可以自動將 DOS 存入記憶體內。如何來分辨您的機器是否爲 AUTOSTART MONITOR呢？當您將APPLE Ⅱ打開，磁碟機發出轉動及磨擦的聲音，而且在磁碟機前板上的" IN USE "燈亮著，那麼您就擁有自動監督程式系統。

當您擁有自動監督程式，但並沒有使用語言系統卡，將DOS 存入記憶體內只用一個步驟就可以完成；將APPLE Ⅱ關掉，將"

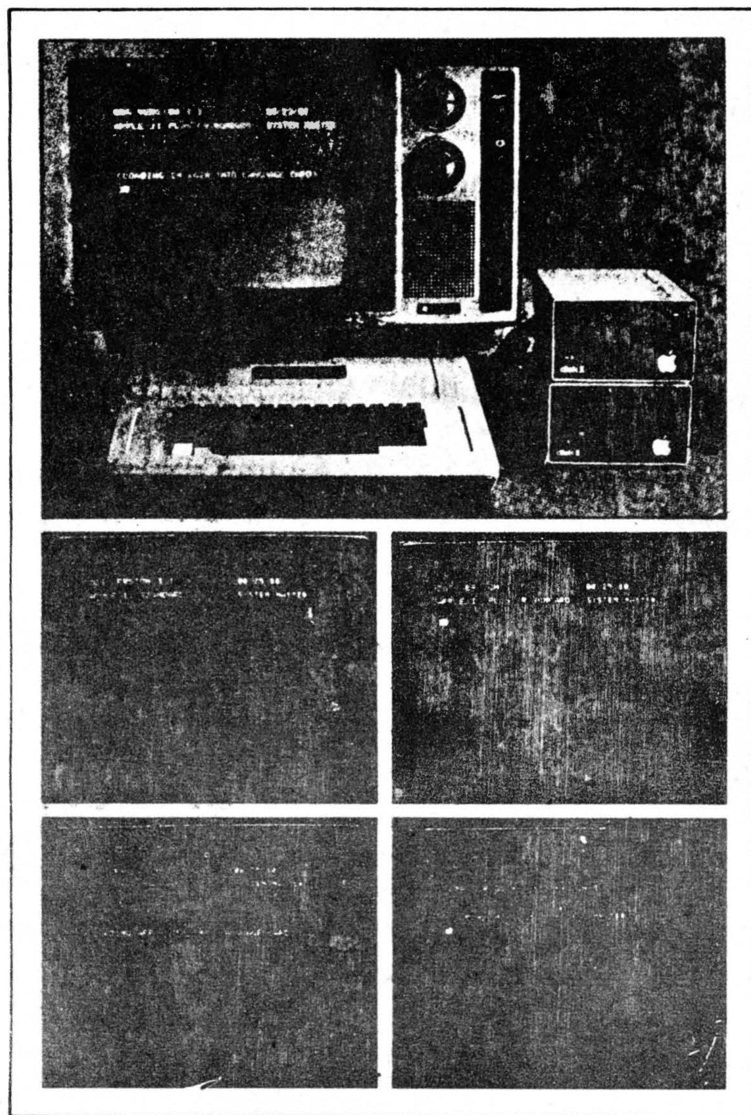


圖2－5　System Master Diskette成功地Boot

SYSTEM MASTER DISKETTE"放入第一個磁碟機內，然後將APPLE Ⅱ的開關打開，幾秒鐘之後，磁碟機將會停止轉動，而螢幕上顯示出如同圖2－5中的一個情形。

從監督程式系統中將DOS存入記憶體內

當"＊"的系統標示出現在螢光幕上時，這就表示監督程式系統等待著您的命令。有幾種方法將DOS從監督程式系統中存入記憶體內。

跳出監督程式系統並將DOS存入記憶體內

您可以使用以下的一個監督程式命令將DOS由第一個磁碟機傳輸到記憶體內：

*C600G

記得要按 RETURN 鍵，磁碟機上的紅燈將會閃亮，過了幾秒鐘之後，螢光幕將顯示如同圖2－5中的一個畫面。

用CTRL-P將DOS由監督程式系統中存入記憶體

我們可以使用另一個監督程式的命令－－CTRL－P將DOS存入記憶體內；使用這個命令，您先將磁碟機所在的擴充接點的數目打入APPLE Ⅱ，然後打入 CTRL－P，再按RETURN鍵；過了幾秒鐘後，您的螢光幕上將顯示如同圖2－5中的一個畫面。

從整數BASIC或APPLESOFT中做BOOTING的動作

在整數BASIC或APPLESOFT 中都是使用同樣的命令將DOS

存入記憶體中。

在BASIC語言系統標示符號（">"爲整數BASIC系統，"]"爲APPLESOFT系統）後，打入IN或PR及一個"#"號，再將要做BOOT 動作的磁碟機所在的擴充接點號碼打入（一般都是6）。指令的型式就如同下面所顯示的其中的一種型式：

```
IN#6
PR#6
```

打入指令後，按 RETURN 鍵，幾秒鐘後螢光幕將顯示如同圖2－5畫面。

從語言系統中將 DOS存入記憶體中

由於 DOS 型式的不同，從語言系統（LANGUAGE SYSTEM）中將 DOS 存入記憶中的方法也就分成兩種不同的型式。

DOS的型式可由"SYSTEM MASTER DISKETTE"的標示上分辨出來，它的型式可分爲3.3，3.2及3.2.1三個版本。

當您的 DOS 型式是 3.3時，不論有沒有語言系統，做BOOT 的動作都是相同的。使用"SYSTEM MASTER DISKETTE"，並遵照上面所提過的步驟，您就可以將 DOS 存入記憶體內；同時整數BASIC及APPLESOFT 兩種語言都可供您選擇使用了。當磁碟機停止轉動時，螢光幕將顯示如同圖2－5 中的一個畫面，由此您也就知道BOOT 的動作已完成了。

當您的DOS型式是 3.2.1 或 3.2 時，做BOOT的動作必須經過兩個步驟；第一步將"INTEGER AND APPLESOFT Ⅱ"的磁碟片放入連接在第六個擴充接點上的第一個磁碟機內，再執行以上所說的BOOT 的動作，磁碟機將會動作，而在磁碟機前板上的"IN USE"燈將會閃亮；幾秒鐘後，您在螢光幕上可以看到下面的訊息：

```
INSERT BASIC DISK AND PRESS RETURN
```

這時您得將"INTEGER AND APPLESOFT Ⅱ"磁碟片拿出來，並將"SYSTEM MASTER DISKETTE"放入，把門關上，按RETURN鍵；幾秒鐘後，您的螢幕也將顯示如同圖2－5中的一種情形，這時DOS已經存入記憶體內了。

如何看磁碟片的目錄

如果您已完成上一節的動作（將DOS存入記憶體內），您也許對磁碟片的內容感到興趣；在APPLE Ⅱ的鍵盤上，您打入：

```
CATALOG
```

這時螢光幕上就會顯示下面的訊息（您是否記得按下RETURN鍵）：

```
DISK VOLUME 254

*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

```
NEW
10    REM INITIALIZED ON date
20    REM SYSTEM MEMORY bytes
30    REM DOS VERSION number
40    PRINT "disk name"
50    END
```

圖 2－6　歡迎程式

如何 BOOTING 其他的磁碟片

我們前面已解釋過如何將 DOS 存入記憶體內；我們可使用同樣的步驟將其他 APPLE Ⅱ 的磁碟片 BOOT 入記憶體內。

磁碟片如果不是爲 DISK Ⅱ 所設計，而是爲其他的電腦或是其他的磁碟機所設計的，那麼這種磁碟片很可能就無法在 DISK Ⅱ 上使用。

如何準備空白磁碟片

當您使用 APPLE Ⅱ 時，您可能需要其他的磁碟片；您若要在 DISK Ⅱ 上使用磁碟片，首先就必須使它變成 DISK Ⅱ 所能接受的型式（即 INITIALIZE）；如果您使用的應用程式(APPLICATI-ON PROGRAM)包含了如何去設定磁片的格式，那您就可以遵照它的指示去做；如果並不包含的話，您也可以遵照以下的指示，做準備的工作。

這種準備的過程（INITIALIZATION PROCESS）使磁

碟片可以隨時被我們使用。在執行這個過程時，儲存在APPLE Ⅱ記憶體內的任何程式就會變成磁碟片上頭的"歡迎程式"（GREETI-NG PROGRAM）。當您 BOOT時，歡迎程式就會自動地存入記憶體內，並且執行；如果您已想好了歡迎程式，則您可以使用它做為GREE-TING PROGRAM，否則您可以按照圖2－6做爲遵循的範例。除了斜體字您必須加入確實資料外，其他的您可以完全不變地使用；DATE 處您加入確實的日期，BYTES 處您填入系統記憶體大小的數目（例如32K或48K等），MUMBER 處您填入您所使用DOS的編號（例如3.2，3.2.1或3.3），DISKETTE NAME處，您可以加入任何您所希望加入的名字。以下所示就是一個範例：

```
NEW
10  REM INITIALIZED ON 1/1/81
20  REM SYSTEM MEMORY 48K
30  REM DOS VERSION 3.2.1
40  PRINT "MISC., VOL. 3"
50  END
```

現在您可以INITIALIZE 磁碟片了，首先將歡迎程式打入機器內，並將磁碟片放到磁碟機中，再打入以下指令：

```
INIT HELLO
```

在您確定磁碟機的門關好後，按 RETURN 鍵。這時磁碟機的紅燈會亮並發出轉動及磨擦的聲音；整個過程將耗費大約兩分鐘的時間，所以您必須耐心地等待。等到紅燈熄滅後，INITIALIZE 的過程就完成了。

現在您可以將磁碟片拿出，並將準備好的標示貼在封套上。

儲存及執行程式

有許多的程式是爲了APPLE Ⅱ而發展出來的，有些存在磁碟片內，有些存在磁帶內，或者兩者都有；有些是用整數BASIC語言設計的，有些是用APPLESOFT 語言設計的；一般說來，用APPLESOFT或整數BASIC 設計的程式只能被用在本身的語言系統內，並不能互相通用。

使用正確的BASIC語言

雖然大多數的APPLE Ⅱ型式都具有兩種型式的BASIC語言系統，但並非所有的都是如此；APPLE Ⅱ PLUS並不具有整數BA-SIC語言系統，除非您在機器上加了一片語言系統卡或整數BASIC語言卡；從另一方面來說，許多的APPLE Ⅱ型式也不具有執行APPLESOFT 語言的能力，除非您從磁帶或磁碟片中將APPLESOFT的編譯程式存入記憶體內。

如果您的APPLE Ⅱ系統內有語言系統卡或APPLESOFT 語言卡，那在您將應用程式存入記憶體內時，機器會自動地選擇適當的語言系統執行您的程式，否則您必須在確定您所擁有的語言系統後，才將您的程式存入記憶體內。

在一個標準的APPLE Ⅱ型式下，您可以很容易得到整數BAS-IC 的系統標示"＞"。您只要按下 RESET 鍵，再按 CTRL－B接著按下 RETURN 鍵，於是您就進入整數BASIC系統了。

但是當您在標準的APPLE Ⅱ內，想要得到APPLESOFT 的系統標示（進入APPLESOFT 語言系統內），那就比較困難了；因爲APPLESOFT 的編譯程式存在磁帶或磁碟中，您必須命令

APPLE Ⅱ將它從磁帶或磁碟內存入記憶體內。

如前節所敍述的，您在使用磁碟機之前，您必須先將DOS存入記憶體內；只要DOS已經被存入記憶體內了，您就可以由"SYSTEM MASTER DISKETTE"內將APPLESOFT的編譯程式存入記憶體中了。首先您將"SYSTEM MASTER DISKETTE"放入DISK Ⅱ磁碟機內，然後打入以下的指令：

FP

按下 RETURN 鍵之後，磁碟機將會轉動並發出磨擦的聲音；幾秒鐘後，APPLESOFT 的系統標示"］"將會出現在您的螢光幕上。

您也可以從磁帶中將APPLESOFT 的編譯程式存入記憶體內；首先拿出標示有"APPLESOFT Ⅱ"的磁帶，將它放入磁帶機內並轉回頭，然後打入以下指令：

LOAD

在您按下 RETURN 鍵之前，首先必須先按下磁帶機上的PLAY鍵，很快地您將聽到APPLE Ⅱ發出"嗶"的一聲，那是表示APPLE Ⅱ 已經開始將APPLESOFT 的編譯程式存入記憶體內了；從磁帶內將APPLESOFT 編譯程式存入記憶體中，大約要花費一分半到兩分鐘的時間，當您聽到又一聲"嗶"的聲音時，您就知道這個過程已經結束了，這時您可以從螢光幕上看到APPLESOFT 的系統標示；在這個時候您才可以停止磁帶機的轉動。

在APPLESOFT 語言系統內，您可以用以下的命令使系統回到整數BASIC內：

INT

這時如果您又想回到APPLESOFT 內，您必須重新由磁碟片或磁帶內將APPLESOFT 的編譯程式存入記憶體中。

從磁帶內將程式存入記憶體中

以下是將程式從磁帶內存入記憶體中的程序（不論您所使用的APPLE Ⅱ是處於那一種語言系統下）：

㈠您先將磁帶轉到程式的開始處；一般說來都是在磁帶的起頭，所以您必須將磁帶完全地轉回到開始的位置才行；如果您所需要的程式並非存在磁帶的起始處，那您必須重覆以下的步驟，先將前面的程式輪流存入記憶體內，直到您成功地將您所需要的程式存入記憶體內爲止。

㈡在APPLE Ⅱ的鍵盤上打入以下的命令：

```
LOAD
```

㈢按下磁帶機上的PLAY 鍵，使磁帶開始轉動。

㈣按下 RETURN 鍵。

在您按下 RETURN 鍵後，游標（CURSOR）將會消失，幾秒鐘之後，APPLE Ⅱ將發出"嗶"的一聲，通知您它已經開始將程式存入記憶體內了，過了一段時間後，您將再聽到"嗶"的聲音，通知您它已完成了這項工作，這時您就可以按下磁碟機上的STOP 鍵，停止磁帶的轉動。

這時程式已經存入記憶體內了，如果您沒有聽到"嗶"的聲音或者在執行儲存動作時，得到錯誤的訊息，您必須依照本章前面所敍述的方法調整磁帶機的音量；如果鍵盤被鎖住了（也就是說鍵盤不接受輸入訊號），您必須按RESET鍵或將機器關掉再打開，如果您打算這樣做，那末您必須重新將APPLESOFT 的編譯程式抄到記憶體中；如果您仍然不能成功地將程式存入記憶體內時（也許您所使用的磁帶已經損壞了）您必須換一捲磁帶。

從磁碟片內將程式存入記憶體中

只要您已經將 DOS 存入記憶體內，您就可以使用以下的命令存入記憶體內：

LOAD *program name* ,

在使用的時候，PROGRAM NAME 處就是您所想要存入記憶體內程式的名字；當然，這個程式名字所代表的程式，必須已經儲存在磁碟片中。

開始執行程式

當您已經將程式存入記憶體內後，可以使用以下的命令使程式執行:

RUN

這時程式控制了整個APPLE Ⅱ的系統（包括了鍵盤及顯示螢光幕）。想要重新得到控制權，在大多數的程式中您可以使用 CTRL -C 再按 RETURN 鍵打斷程式的執行，如果並不生效，則您可以參看這個程式的操作命令。如果您處在緊急狀況下，希望能立即得到機器的控制權，您可以按 RESET 鍵或關掉機器再打開，但在這種情形下，您將可能失去記憶體中的程式。

調整顯示螢光幕的彩色顯示

APPLE Ⅱ 能夠顯示所有的彩色圖形；如果您所使用的程式想要利用這一項特色，您首先必須先調整顯示螢光幕或電視機的色彩，使彩色達到正確的顯示，APPLE 電腦公司提供了一個程式來從事這方面的

工作；在整數**BASIC**語言系統下，使用"**COLOR GRAPHICS**"磁帶，而在**APPLESOFT**語言系統下，使用"**COLOR DEMO-SOFT**"磁帶或存在磁碟片上的**COLOR DEMOSOFT**的程式，您可以得到如同圖2－7所顯示的螢光幕。

圖2－7所顯示的情形，被稱爲"手册"（**MENU**）。**MENU**提供了數種的格式（圖2－7顯示4種格式），供您選擇。在確定顯示螢光幕的彩色時，您選擇第一種格式（打入1）並按下 **RETURN** 鍵；這時螢光幕將顯示16種顏色（如圖2－8所顯示的情形）。

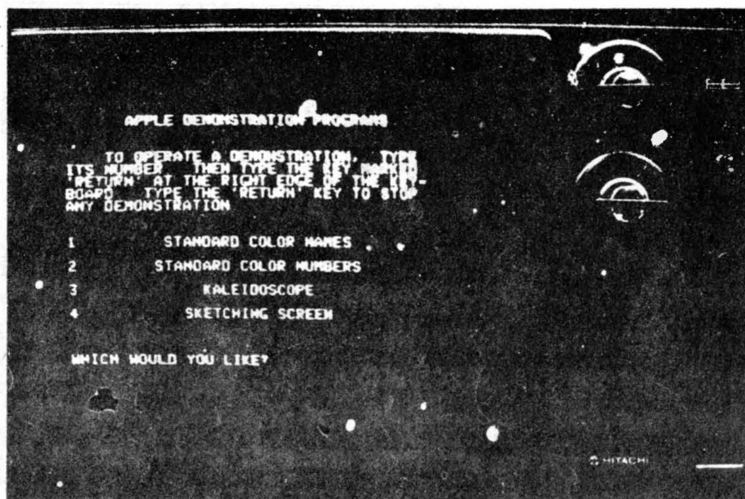


圖2－7　COLOR DEMOSOFT程式手册

如同您所見到的，在螢光幕的下方搭配著該種顏色的名字；從左到右，這些顏色分別是：

BLAK	Black	BROWN	Brown
MGTA	Magenta	ORNG	Orange
DBLU	Dark Blue	GREY	Grey
PURP	Purple	PINK	Pink
DGRN	Dark Green	LGRN	Light Green
GREY	Grey	YELO	Yellow
MBLU	Medium Blue	AQUA	Aqua
LBLU	Light Blue	WITE	White

現在您可以調整對比、亮度、色彩及色度，直到您認爲可以接受時爲止；在調整的時候，必須特別注意紫色、粉紅色、黃色及三種藍色。

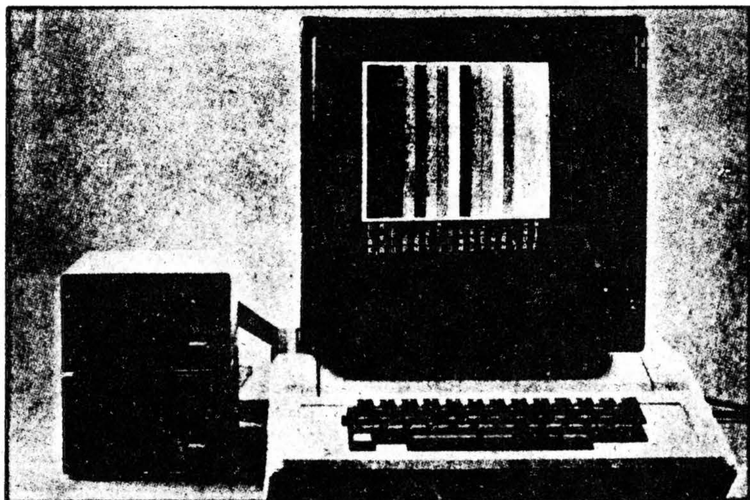


圖2－8　調整TV顏色

待您結束了調整色彩的工作後，您可以按 RETURN 鍵回到MENU的地方，這時您可以選擇一個項目做一下色彩的試驗。若您想要結束程式時，您只要打入 CTRL －C 並按下 RETURN 鍵就可以了。

各式各樣的配件

各種系統設備的操作指令都已經在前面詳述過了，至於其他的配件，您必須參考該項配件所提供的參考手册，遵循它的步驟，從事操作的程序。例如您擁有一套列表機，那麼您必須遵照該套列表機的操作手册，從事操作的工作，因為列表機的種類太多，它們所提供的功能或操作方式都有很大的不同，並非只有一種操作程序。

錯誤的顯示

雖然APPLE Ⅱ是一個非常神奇的機器，但是它也有一個所有電腦共同具有的問題，那就是它缺少想像的能力。您所打入的命令，必須完全正確，否則它將不會如您所期望地工作。一個錯誤的命令，可能由最輕微的損害，導至非常嚴重的後果。

錯誤訊息

當您在APPLE Ⅱ上打入錯誤的命令並按下 RETURN 鍵，這時APPLE Ⅱ往往會發出"嗶"的聲音，並在螢光幕上顯示出錯誤訊息（ERROR MESSAGE）；一般說來，錯誤訊息將會給您一個指引，告訴您的錯誤在那裏，但是有些情況並不能給您這個指引。最簡單的修正方法就是重新打入這個指令。在附錄C中您可以看到所有的錯誤訊息。

修正打錯的命令

當您在APPLE Ⅱ的鍵盤上打入命令時，您可能會發生一些錯誤；在您按 RETURN 鍵之前，如果您發現所打入的指令有不正確的地方，這時您可以利用前面所描述的一些鍵做修正的工作，它們分別是：←鍵，→鍵，REPT 鍵，CTRL -X 命令及 ESC -@ 命令，現在依照順序分別描述如下：

- ←鍵使游標往左邊移動，並消除所有經過的字，雖然所經過的字仍然顯示在螢光幕上，但它們已經在記憶體中被洗掉了。
- →鍵使游標往右邊移動，並將所經過的字記憶在記憶體中。
- REPT 鍵與→鍵或←鍵合用時，分別使記憶的工作及消除的工作得以快速執行。
- CTRL -X命令使得您剛才所打入的字完全消失。
- ESC -@命令消除整個螢幕的顯示，並將游標移到螢幕左上角處（即母位）。

現在讓我們看看如何應用這些修改的特性。假設您想打入下面的命令：

```
LOAD COLOR DEMOSOFT
```

結果您打入了下面的錯誤命令，但在按 RETURN 之前您已發現到這個錯誤：

```
LOAS COLOR DEMOSOFT
```

這時您有兩個選擇：第一您可以打入 CTRL -X取消整列的資料，然後重新開始；或者您可以使用←鍵回到錯誤的地方，再加以改正；您

可以按住←鍵，再按 REPT 鍵，讓游標回到這行的起點。當游標回到發生錯誤的地方時，放掉 REPT 鍵，因爲當游標往左移動時，它所經過的字都將在記憶體內消失；當游標往左移動得過度時，您必須使用→鍵使游標位在S字上，這時您就可以打入正確的字——D了，這時整行看起來都是正確的，但您知道如果您現在按下 RETURN 鍵的話會有什麼情況發生嗎？

LOAD COLOR DEMOSOFT

這時在游標右邊的所有字母將會消失無蹤。

LOAD

這種情形並非您所希望輸入的資料，所以在您按 RETURN 之前，您必須將游標移到這一行結束的地方。您當然也可以重打一遍，但您可能重新又造成了另一個錯誤，使用→鍵（與 REPT 鍵聯合使用）將游標移到這一行結束的地方，自動地將所經過的字輸入記憶體內。

不經意的 RESET

在您並不想要按 RESET 鍵時，您很可能不經意地按到了它；除非您所使用的機器必須使用 CTRL－RESET 才能使 RESET 鍵發生作用，否則它將造成很大的損失。您可以將 RESET 鍵上的塑膠蓋拿掉，只剩下鍵內的支撐柱，這時您可以取一個內圓直徑⅜英吋，外圓直徑½英吋及厚⅛英吋的塑膠環套入支撐柱上（如同圖2－9及2－10所示的情形），再將塑膠套套上；這時您就使得 RESET 鍵被保護住了，只有用力的按下 RESET 鍵才能使它發生作用。



圖2－9 防止不經意的碰觸 Reset ：第一圖



圖2－10 防止不經意的碰觸 Reset ：第二圖

從不經意的 RESET 恢復工作

當您不經意的按下 RESET 鍵後，您所能做的事完全依賴您所正在做的事及您所使用APPLE Ⅱ的型式。

您所使用的APPLE Ⅱ的程式應該都有它們的指令，告訴您在您不小心按下 RESET 鍵時所應該做的事，所以當您執行這個程式的時候，您必須確定您知道如何去處理這種情況；當您在執行一個BASIC程式時，不經意地按下 RESET 鍵，將使整個程式重新開始執行，這似乎看起來並不嚴重，但是當您在執行一個會計程式或類似的應用程式時，不經意地 RESET 可能將整個事情弄得一團糟。

當您按RESET 鍵後，APPLE Ⅱ將停止所有它正在執行的工作，控制權回到鍵盤，您將看到系統標示的符號及游標出現螢光幕的下方閃動；如果您處於BASIC系統內，您可以打入 RUN 指令，重新執行這個程式，但是您剛才執行程式所產生的結果可能都會消失。

如果您看到監督程式系統標示" * "時，您可以使CTRL -C回到BASIC系統（只要您的APPLESOFT 系統不是經由磁碟或磁帶所建立的）。從監督程式系統回到由磁碟片進入的APPLESOFT 系統，您必須打入以下的監督程式命令：

```
*3D0G
```

從監督程式系統回到由磁帶進入的APPLESOFT 系統，則您必須打入以下的命令：

```
*0G
```

小心： 當您打入以上的指令時，您必須有十分的把握，否則請問問別人，因爲如果您打入錯誤的指令，您必須重新將您的程式、APPLESOFT語言系統或 DOS 存入記憶體內。

3



如何用BASIC語言寫程式

這一章將告訴您如何開始使用**BASIC**語言來設計**APPLE II**的程式。

BASIC是一種與其他程式語言具有相似功能的程式語言，它具有許多的指令可以由您組合成爲程式，而一個程式也就是您希望電腦爲您做什麼樣工作的工具。

我們可以經由強迫您去學習**BASIC**指令而使您學會**BASIC**語言，但是由於個別的指令並不具有什麼實際的意義，您可能因此而放棄；如果您由指令開始學習，那麼往往您所得到的只是一大串規則，而不能得到任何設計程式的技巧及練習設計程式的機會，因此完整的**BASIC**指令規則，您將可以在第八章中發現；如果您需要瞭解個別指令的明確定義，您可以參考第八章，但千萬不要在看完這一章之前做這樣的事。

BASIC的第一步

APPLE II具有兩種**BASIC**語言的能力，有些型式能夠執行整數**BASIC**語言，有些能夠執行**APPLESOFT**語言，也有一些能夠

執行這兩種語言。我們將在後面詳述這兩種語言不同的地方，所以在現在您不必憂慮您所使用的是那一種BASIC語言。

從BASIC的系統標示開始

APPLE II 是一個具有多種語言能力的電腦，如果您希望用BASIC語言設計程式，那麼它必定期待BASIC的指令；您將可以在本章中看到如何開始使用BASIC指令，現在讓我們很快地看下去。

如果您的APPLE II 擁有自動監督程式，您只須將電源打開，然後按下 RESET 鍵，那麼您就進入BASIC 語言系統了。

如果您的APPLE II 並不具有自動監督程式，那麼您就必須將電源打開，然後按CTRL-B，再按RETURN鍵。

在螢幕左邊的系統標示告訴我們的系統是那一種BASIC語言；如果您看到的是“ > ”，那麼您就擁有整數BASIC語言系統，如果是“] ”，則您就在APPLESOFT 語言系統中了。

直接式執行及間接式執行

在我們開始擔憂如何在APPLE II 上變換整數BASIC及APPLESOFT 之前，讓我們先看一些能在 APPLESOFT 及整數BASIC上使用的指令。在本章中有些例子使用整數BASIC的系統標示“ > ”，有些則使用 APPLESOFT 的系統標示“] ”，但不論您看到的是那一種系統標示，它們在這兩種系統中都可以正確地被執行（除非有其他特別的說明）。

印出字母

當您進入BASIC語言系統時，您就處在立即式（IMMEDIATE

MODE)的狀況下——也被稱爲直接式(DIRECT MODE)或計算式(CACULATOR MODE)。在這種狀況下，APPLE II對您所打入的指令會立刻作反應；試著打入以下的敍述：

```
PRINT "LET SLEEPING DOGS LIE"
```

在您打入這一行後，別忘了按 RETURN 鍵，這時APPLE II將會印出下面的訊息：

```
LET SLEEPING DOGS LIE
```

如果APPLE II印出"? SYNTAX ERROR"或"*** SYNTAX ERR"的訊息，那就是告訴您，您的命令它不能瞭解，您可能拼錯了PRINT這個字。如果APPLE II只印出0而並非其他的訊息，那就表示您忘了打入第一個引號。不論是以上的那一種情形發生，您都可以重新打入正確的敍述，但是要小心一些，以免再犯錯誤。電腦是一絲不苟的東西，不論是一個字或標點符號的錯誤，都將使它受到妨碍，或者產生更糟的後果。

一個如同上面的命令，使得電腦在螢光幕上印出兩個引號中間的所有資料。

在兩個引號之間放入的訊息長度有一個極限，而這極限在整數BASIC及APPLESOFT間並不相同，但都超過了螢光幕一行所能顯示的長度，這就表示一個命令可以超過螢光幕一行所能顯示的長度。像這樣的一個較長的命令將會自動地被移到下一行的位置去。試著打入以下的命令：

```
PRINT "UNDER NORMAL CIRCUMSTANCES, THE M
AN WOULD BE CONSIDERED CRAZY"
UNDER NORMAL CIRCUMSTANCES, THE MAN WOUL
D BE CONSIDEREB CRAZY
```

整數BASIC允許一個命令的長度極限是120個字，如果您超過了這個極限，在您按下 RETURN 鍵之後，您將得到"***TOO LONG

ERR ”的訊息。

APPLESOFT 允許您輸入255個字的長度，當您的輸入長度接近255的時候，APPLE Ⅱ 將發出一連串“嗶”的聲音，當輸入超過255個字的時候，APPLE Ⅱ 將印出“＼”的符號，並自動地取消您的輸入，就如同您按了C_{TRL}－X一般。

印出計算結果

您可以使用APPLE Ⅱ 的立即式去執行計算的工作，就如同您使用小計算機（CALCULATOR）一般，它將即刻把計算的結果顯示出來。您不妨試試以下的例子：

```
PRINT 4+6              加    法
10

PRINT 500-437          減    法
63

PRINT 100*23           乘    法
2300

PRINT 96/12            除    法
8

PRINT 3^2              指數計算
9

PRINT 3*4*10-800       混合計算
-680
```

在每個命令的下面就是正確的答案，您是否注意到在這些例子中我們都沒有用到引號？您知道如果您使用了引號之後會有什麼樣的結果嗎？如果您並不確定，那麼您可以試試看。

整數BASIC在作計算時，數值（NUMERIC VALUES ）有一個上下限，如果在運算時所得到的值超過了32767，那麼您將得到“*** > 32767 ERR ”的訊息，反之若小於－32767，則您得到的將是“－*** > 32767 ERR ”的訊息。以下是幾個錯誤的例子，最後一個是用0做除數的錯誤例子：

```
>PRINT -32766-2
-*** >32767 ERR
>PRINT 2^15-1
*** >32767 ERR
>PRINT10/0
*** >32767 ERR
```

小數點以下的位數在整數BASIC中並不存在，所以當您做除法時，若不能整除，那麼小數部份就會被刪掉。例如以下的例子：

```
>PRINT 9/2
4
```

APPLESOFT 允許小數的存在，一個數值可以有九位有效數字（包含整數及小數部位），這表示一個超過九位的數值將被四捨五入成爲九位或更少的有效位數；以下就是一些例子：

```
]PRINT 12.34567896         五　入
12.345679

]PRINT 12.34567894         四　捨
12.3456789
```

如果您使用 APPLESOFT 立即式計算一些您的算術計算式子，您將發現在某些時候您會得到科學表示（SCIENTIFIC NOTATION）的結果。

```
]PRINT 123456789123
1.23456789E+11
```

如果您不瞭解所謂的科學表示，那麼您目前只要做一般的算術運算，我們將在這一章的後面討論所謂的科學表示及數值。

PRINT敍述的簡寫

APPLESOFT 允許您使用“？”代替PRINT指令，以下是一

些例子：

```
]?"TIME MARCHES ON"
TIME MARCHES ON

]?13-46*6
-263
```

錯誤訊息

到目前爲止我們已提過了幾種當APPLE Ⅱ測試出它所不能接受的命令時所給的錯誤訊息，當它測試到錯誤時，它將發出“嗶”的聲音吸引您的注意力，並設法爲您偵測出錯誤的所在；由於它的偵錯能力有限，所以您不必期望它會給您一個確定的錯誤分析。目前APPLE Ⅱ本身只有大約 35 種不同的錯誤訊息，它們可以大致的告訴您錯誤的所在以及是怎麼樣的錯誤。

在這一章及本書的其他章節中，我們將對某些特定的錯誤所產生的錯誤訊息作一個說明，尤其是一些不太容易發現的錯誤。在附錄C中您將發現按照字母順序排列的所有錯誤訊息。

錯誤訊息的格式

在整數BASIC中及APPLESOFT中，錯誤訊息的格式有些不同，如同以下所顯示的：

```
]PRNIT "THE LAVA FLOWS"

?SYNTAX ERROR
]

PRNIT "THE LAVA FLOWS"
*** SYNTAX ERR
>
```

額外的空格

您是否正在為BARIC敍述中何處應該留空白或何處不留空白而煩惱？這一點您不必多慮，因為APPLE Ⅱ 編譯BASIC指令只看其中的組成元素（ELEMENTS），空白是不考慮的；所以您所需要考慮空白的地方只有在引號之內，因為在引號之內的空白也被認為是敍述的一部份。

敍述、行及程式

BASIC敍述的功能是由APPLE Ⅱ 提供的，對於它所能做的工作具有明確及完整的定義，而一個程式就是由許多的敍述組合而成的；如果一件工作是簡易的，那麼程式就可能是簡易的；像立即式的情形就是一個最簡單的例子，它僅由一個敍述組合而成，對APPLE Ⅱ 而言只是一個指令。這是一種特殊的情形，一般的程式往往具有10個、100個、1000個或更多的敍述。您看以下的敍述：

```
PRINT "COWS MOO"
COWS MOO

PRINT "FOR FANCY BLUE"
FOR FANCY BLUE

PRINT "HOOF-B-NU"
HOOF-B-NU
```

每一個在立即式之下執行的程式在螢光幕上印出一行字（TEXT）的訊息，每個程式只有一個敍述及一行位置。

APPLESOFT 允許您在同一行內寫上不只一個的敍述，您只要使用冒號“：”在同一行中將不同的敍述分開就行了，您不妨用下面的例子與前面的許多例子做一個比較：

```
]PRINT "COWS MOO":PRINT "FOR FANCY BLUE"
:PRINT"HOOF-B-NU"
```

```
COWS MOO
FOR FANCY BLUE
HOOF-B-NU
```

這一行中包含了三個敍述，但它的結果與分開成三行敍述所顯示的結果相同。

在 APPLESOFT 中一行所能包含的敍述的數目並沒有限制，但您必須注意的是，一行中所能包含的長度只有255個字；如果您打入一長串的字，當您在打入第248個字時APPLE Ⅱ 將會開始發出"嗶"的聲音，警告您已經在接近這個極限了。如果您超過了255個字，APPLE Ⅱ 將自動地取消這一行的輸入，就如同您打入CTRL－X一般，於是您就必須重新做輸入的工作。APPLE Ⅱ 將不會執行任何過長的程式行中的指令，所以當您在立即式的情況時，對您所能做的事就有了一個限制。

APPLESOFT中的一行程式

您可以在一行中放入許多的指令，這就必須感謝 APPLESOFT 能夠處理一行255個字的能力了；例如，您考慮以下的敍述：

```
]FOR I=1 TO 800:?"A";:NEXT:?"PHEW!"
```

不管這個小程式的意義，您只管打入電腦中，並按下 RETURN 鍵，如果您打的正確，您將發現字母"A"在螢光幕上顯示了20列（每列40字）之多，並且在第21行上顯示了"PHEW！"。

```
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
PHEW!
```

這時這一行程式留在螢光幕的上方，這是由於這個程式執行後只使程式的顯示移到最上一行，而不會使它移出螢光幕外而消失。

當這一行程式執行完畢後，**APPLESOFT** 的系統標示顯示在游標的左方——位於螢光幕的最後一行。

間接式的執行

到現在爲止我們所做的程式都是教育性質的，希望它們也是有趣的，但在立即式的情況下，您所能做的工作是非常有限的；這在整數 **BASIC** 而言更是如此，因爲它規定每一行只能包含一個敍述。在立即式的情況下，另外一個麻煩就是在您想要使用它時，您必須重新打入重複的指令；當然在 **APPLE Ⅱ** 上可以使用一些編輯（**EDITING**）的能力使得在螢幕上顯示的程式能夠被重複使用，但是仍然受到限制。

您所需要的是能夠輸入許多的程式行（**PROGRAM LINES**），而且能夠重複地使用它們；這與只有一行的程式相比，就顯得比較複雜了。

解決立即式程式問題的方法，就是使用程式執行（**PROGRAM**-

MED MODE)的方式來撰寫程式，這種方法也被稱爲間接式(IND-IRECT MODE)。在間接式的狀況下，電腦將您輸入的程式存在記憶體內，但並不執行程式中的任何命令，直到您告訴它開始執行時爲止。您可以依您的需要，輸入任何數目的程式行，然後在適當的時機，告訴電腦執行您的程式，這時電腦才會依照您程式中的命令執行您的程式。

程式的執行

我們說電腦執行(EXECUTES)或跑(RUNS)一個程式，那就是說電腦按照程式的命令去做一項工作。

在立即式狀況時，當您按下 RETURN 鍵，電腦就執行您的程式。

在間接式狀況下，您必須打入 RUN 這個指令，才能使電腦執行您的程式。每當您打入 RUN 一次，程式就被電腦執行一次。

清除記憶體中的程式

由於APPLE II 將您的程式存放在記憶體中，因此在您重新輸入一個程式之前，您必須先將原先儲存在記憶體中的程式消除。您只須打入NEW指令就可以完成這項工作；如果您忘了打入NEW指令，那麼原先的程式就會與您新打入的程式混在一起了。

適當地結束程式

在立即式的情況時，程式執行的結束是顯而易見的；但在間接式的情況下却並不如此；我們使用 END 指令來結束程式的執行，所以在您的程式中，END 指令應該是您程式的最後一個指令。

不過，在 APPLESOFT 中並不需要 END 指令做爲程式的結束，它可以在所有的指令執行完畢後，自動地結束程式的執行。

行　號

在間接式的狀況下，行號（LINE NUMBERS）是最重要的；行號只是簡單的一個、兩個、三個、四個或五個數字，位於程式行的起頭。直接式與間接式的差異，就在於有沒有行號的存在。有一些指令，只可以被使用在立即式的狀況下，而又有一些指令只能用在間接式，這一點我們將在稍後討論。

試著執行以下的一個程式：

```
>NEW

>10 PRINT "RUBBER BABY BUGGY BUMPERS"
>20 END
>RUN
RUBBER BABY BUGGY BUMPERS
```

行號必須是獨一的，任何兩個程式行絕不可以擁有同樣的行號；如果您在不同的程式行中使用同樣的行號，那麼電腦將只記得最後輸入的程式行。打入以下的程式，您就可以明瞭上面所說的意義了：

```
>NEW

>10 PRINT "FIRST LINE 10"
>10 PRINT "SECOND LINE 10"
>20 END
>RUN
SECOND LINE 10
```

在BASIC程式中，行號決定程式行執行的先後順序；第一個程式行必須擁有數目最小的行號，而最後一個程式行則必須擁有最大數目的行號；即使您打入的程式行的行號並不依照順序排列，APPLE Ⅱ 也會按照行號的大小在電腦內替您依序排列完成。參考以下的一個行號沒有按照順序排列的程式：

```
>NEW

>30 PRINT "CUT"
>10 PRINT "FISH"
>20 PRINT "OR"
>40 PRINT "BAIT"
>50 END
>RUN
FISH
OR
CUT
BAIT
```

我們使用 Esc－@ 命令清除畫面，再重新執行上面的程式，我們可以證明APPLE Ⅱ 並沒有忘掉間接式的程式。

```
> ── 按 Esc-@，再按 RETURN

>RUN
FISH
OR
CUT
BAIT
```

將存在記憶體中的程式加入程式行是一件簡易的事，您可以在程式的起頭，結尾或程式的任何地方加入擁有適當行號的敍述。假設您想在上面那個例子中在程式的開頭加入一個程式行，您可以加入以下的程式行就行了：

```
>5 PRINT "EITHER"
>RUN
EITHER
FISH
OR
CUT
BAIT
```

因爲前面例子的第一個程式行的行號是 10 ，所以您加入的程式行的行號，只需小於 10 就可以把這一個程式行加在程式開頭了。

這個程式的行號由 10 開始編起，而不是由 0 開始，這是一個好習慣；希望您將來在設計程式的時候，能夠選擇一個適當的起始行號，並在每兩個程式行的行號中預留足夠的數目，這樣您才可以在以後加入程式行時減少困難。

多重敍述的程式行

您可以在一個程式行中放入許多個敍述，第一個敍述緊跟著行號，第二個敍述緊跟著第一個敍述，但中間必須加上冒號“：”；我們使用冒號“：”分開同一程式行中的不同敍述。

整數BASIC語言允許您在間接式的狀況下，在同一行中使用不同的敍述，不像在立即式情況下只能使用一個敍述。程式行所能容納的字數的限制爲150個字左右，確實的長度則依賴該行所包含的內容。如果您打入太長的一行，則APPLE Ⅱ將會給您一個 *** TOO LONG ERR 的錯誤訊息，而您必須重新打這一行。

在 APPLESOFT 中，不論在立即式或間接式的狀況下，一個程式行中都可以包含許多的敍述，在這兩種情形下，一行所能包含字的數目爲255，就如同前面我們所描述的一樣。

列出程式行

您可以打入LIST命令，把存在電腦記憶體內的程式列顯示出來；您現在就可以試試看，如果您在打入NEW指令之前，打入LIST指令，那麼您應該在螢光幕上看到以下的顯示情形：

```
LIST
  5 PRINT "EITHER"
 10 PRINT "FISH"
 20 PRINT "OR"
 30 PRINT "CUT"
 40 PRINT "BAIT"
 50 END
```

這就被稱爲程式列印（PROGRAM LISTING）；LIST命令有許多不同的變化，可以讓您列印一行或是一組的程式行，後面的這種功能使得您能夠適當地應用螢光幕顯示的能力，而不致於受到程式太長的困擾；您現在打入以下的指令，您將可以看到上面那個程式的顯示情

形：

```
LIST 10
```

您看到的是：

```
10 PRINT "FISH"
```

如果您想要列印一串連續的程式行，那麼您必須指出起始及結尾的行號，就如同以下的例子：

```
LIST 20,40
   20 PRINT "OR"
   30 PRINT "CUT"
   40 PRINT "BAIT"
```

在 APPLESOFT 中，您可以列印出某一行號以上的所有程式行（包含這一行號的程式行在內），您也可以同樣地列印出某一行以下的所有程式行（包含這一程式行）。以下就是這兩種情形的例子：

```
]LIST,10

5  PRINT "EITHER"
10  PRINT "FISH"

]LIST 30,

30  PRINT "CUT"
40  PRINT "BAIT"
50  END
```

打斷列印的動作

您可以使用 C_{TRL} - C使得列印的動作中途停止，這是一個用來中斷一長串程式列印的最有效方法。

如果您的 APPLE Ⅱ 擁有自動監督程式，那麼您可以使用 C_{TRL} - S使得程式的列印動作暫時停止，當您按下空白鍵（ SPACE BAR ）時，程式將繼續列印。 C_{TRL} - S使得您可以隨時觀看某一段的程式

而不會有遺漏的情形。

行號的自動增加

APPLE Ⅱ 的整數BASIC語言系統具有自動編出行號的功能；您可以使用 AUTO 指令命令它從事這個工作，當您按下 RETURN 鍵之後，APPLE II 自動地提供下一個行號，但是如果在目前的程式行中有錯誤存在，那麼它將不提供下一個行號，同樣的，如果您並沒有輸入任何的敍述，那麼APPLE Ⅱ 也不提供下一個行號。

您可以使用CTRL-X取消APPLE Ⅱ 自動編行號的功能，接著您打入 MAN 指令，告訴它您想要自己訂行號。以下就是 AUTO 及 MAN 指令的使用例子：

```
>AUTO 1000

>1000 PRINT "HOW MANY YARDS IN A MILE?"
>1010 按 RETURN 鍵
>1010 PRIMT 5280/3
*** SYNTAX ERR
>1010 PRINT 5280/3
>1020 \   按 CTRL - X鍵
>MAN

>
```

如同例子所示，您可以發現使用 AUTO 指令，您必須告訴APPLE II 行號自動增加的起始數目，您當然也可以同時告訴它增加的間隔數目。以下就是一個例子：

```
>AUTO 1000,100

>1000 PRINT "FISH OR CUT BAIT"
>1100 \按 CTRL - X 鍵
>MAN
```

在這個例子中，行號的增加是以100做爲單位的，如果您不告訴APPLE Ⅱ 增加的數目，那麼它將假設以 10 爲單位累進。

APPLESOFT 並不提供自動增加行號的功能。

將程式存在磁帶上

磁帶機提供您一個將程式放入外在器材上及再由該器材存入記憶體內的工具。假設您有以下的一個程式存在記憶體內：

```
10 PRINT "TOTO,"
20 PRINT "I"
30 PRINT "DON'T"
40 PRINT "THINK"
50 PRINT "WE'RE"
60 PRINT "IN"
70 PRINT "KANSAS"
80 PRINT "ANYMORE"
90 END
```

想要將這程式存在磁帶上，您必須將磁帶放入磁帶機內，再將磁帶轉回到起始的地方，然後同時按下磁帶機上的 PLAY 及RECORD鍵，再由鍵盤上輸入以下的命令：

```
SAVE
```

APPLE Ⅱ 將在開始記錄時發出"嗶"的一聲，等到記錄完畢時再發出"嗶"的一聲，這時您就可以按下磁帶機上的 STOP 鍵了。

這時候您可以打入NEW指令，消除記憶體中的程式，再用LIST指令證明記憶體中已經不存在任何東西了。接著您不妨將磁帶捲回起頭處，按下磁帶機上的 PLAY 鍵，再輸入以下的指令，您就可以將程式存入記憶體內了：

```
LOAD
```

APPLE Ⅱ 在開始將磁帶上的程式存入記憶體時，同樣地發出"嗶"的一聲，而在結束時，也做同樣的動作。當您聽到第二聲"嗶"的聲音後，您必須停止磁帶的轉動，這時您就可以使用LIST指令證明程式已經存在記憶體中了。

在第五章中我們將介紹如何將程式存入磁碟中，再由磁碟中將程式存入記憶體內的方法；比較起來，這是要比磁帶來得方便得多的一種方法。

將許多的程式存入磁帶內

您也許注意到，將一個程式存入磁帶內，並不佔用太多的位置；當然，您的程式愈長所佔用的磁帶位置也愈大，但是一般說來，一捲磁帶都具有足夠的空間同時儲存幾個BASIC程式；您可以按順序將程式存入磁帶內：第二個接著第一個，第三個接著第二個。

將第二個、第三個或接下去的程式由磁帶存入記憶體內，並不像存第一個程式那麼方便；在您將磁帶捲回起頭後，您想存入第二個程式時，您必須先經過第一個程式，同樣地，您想存入第三個程式，必須先經過第一及第二個程式，才可以將您想要的程式存入記憶體內。您可以使用下面的方法達成這個目的，您可以重複地使用 LOAD 命令，直到您想要的程式已經被存入記憶體內為止；這是一個有效但很緩慢的方法。

如果您的磁帶機具有記數的功能，那麼您可以增快以上的工作；在您將磁帶轉回頭時，將記數器定在 ∅ 的數目上，在您由記憶體內將第一個程式存入磁帶後，記數器顯示的數目就是第二個程式的起始位置，同樣的，第二個程式結束後記數器的數目就是第三個程式的起始位置。

現在您想要將第二個程式存入記憶體時，您將磁帶轉回頭，記數器定在 ∅，然後使用磁帶機上的FAST FORWARD 鍵，使得磁帶轉到第二個程式起頭的地方，如果您使磁帶轉過了頭，那麼您也可以使用REWIND鍵，使磁帶回頭；這時您就可以使用 LOAD 指令將程式存入記憶體內了。

BASIC語言系統的調換

在APPLE Ⅱ的許多型式中，您可以在 APPLESOFT 或整數 BASIC 語言系統中自由選擇您想要使用的語言系統。在您看完這一章後您將發現選擇某一種BASIC 語言系統的原因；例如：我們在前面看到整數BASIC 具有自動增加行號的功能，而 APPLESOFT 則並沒有，在另一方面來說， APPLESOFT 准許您使用帶有小數的數值，而整數BASIC 則不能容忍。

雖然大多數的APPLE II 型式具備有兩種BASIC 語言的能力，但並非所有的都是如此；例如，標準的 APPLE II PLUS只具備了 APPLESOFT 語言的能力。由一種BASIC 語言轉入另一種BASIC 語言的過程也因爲機器型式的不同或您所加入的介面板不同而有所差異。

如果您的機器加了語言系統卡，那麼您可以任意地使用任一種 BASIC 語言系統；如果您處於整數BASIC 語言中，您只需使用FP 指令就可以轉換成 APPLESOFT 語言系統了；反之，您打入 INT 指令，就可以使您進入整數BASIC 語言系統。

若您使用了 APPLESOFT 語言卡，您也可以自由地使用任一種 BASIC 語言，這片卡上有一個開關，可以供您做選擇，如果您將這個開關壓下，那麼您就進入整數BASIC 系統，若您將開關扳上，則您可以使用的語言就是 APPLESOFT ；同樣地，您可以使用FP 或INT 指令在整數BASIC 及 APPLESOFT 中自由選擇適當的語言系統。

在標準的APPLE II 機型上，您可以按下 RESET 鍵，再打入 CTRL -B 而輕易地得到整數BASIC ；但由於標準APPLE II 的 APPLESOFT 語言系統存在磁帶或磁碟片上，要使用 APPLESOFT 就較爲困難了，您必須先從磁帶或磁碟片中將APPLESOFT 存入記憶體內，才可以使用它的指令。

在您使用磁碟機之前，您必須先將 DOS 存入記憶體內（ BOOT 的動作如第二章所述）；您在每次開機之後都必須做這個動作；現在我們大致地溫習一下在監督系統及整數BASIC 中做這個動作的指令：

*6.　按 CTRL-P, 再按 RETURN

>PR#6

只要 DOS 被存入記憶體中了，您就可以將 APPLESOFT 的磁碟片放入磁碟機內，再打入 FP 指令，幾秒鐘後您將看到 APPLESOFT 的系統標示“] ”顯示在螢光幕上，那時您就已經從整數 BASIC 進入 APPLESOFT 語言系統了。

當然您也可以從磁帶中得到 APPLESOFT ，詳細的步驟可以參考第二章；大致的情形是，您將 APPLESOFT 的磁帶放入磁帶機中，壓下磁帶機的 PLAY 鍵，再由整數BASIC 系統中打入 LOAD 指令；大約 1 ½ 或 2 分鐘內，您可以在螢光幕上看到 APPLESOFT 版權所有的文字及系統標示的符號。

在 APPLESOFT 系統中，您可以使用 INT 指令回到整數 BASIC 系統中，如果您又想回到 APPLESOFT 系統中，而您又擁有磁碟機，那麼您只須打入 FP 指令即可，若您只擁有磁帶機，那麼您就只有使用 LOAD 指令了。

較高超的編輯技術

在第二章中我們介紹了在按下RETURN鍵之前改正一個程式行的方法，現在讓我們很快地回憶一下那些編輯的技術：

←鍵使游標往左移動，同時並將它經過的字自記憶體中消除；雖然程式行中的字仍然顯現在螢光幕上，但是實際上已經不存在了。

→鍵使游標往右移動，同時並將所經過的字存入記憶體內，就如同重新打過一般。

REPT 鍵與←鍵或→鍵合用時，使得前面兩項的功能執行加快。

CTRL -X 命令使得您剛打入的程式行被取消。

ESC - @ 命令將螢光幕上的所有資料消除，同時並將游標移往左上角（母位）。

現在我們將介紹一些編輯程式行的新方法，這些方法在您使用間接式（**PROGRAMMED MODE**）時特別有效。

消除程式行

想要消除某一個程式行，您只需打入該行的行號然後按下RETURN鍵即可。當您重新列出這個程式時，您將可以發現這個程式行已經不再在這個程式中了；以下就是一個例子：

```
>NEW

>100 PRINT "VIRTUE IS ITS OWN REWARD"
>110 PRINT "IF THE SHOE FITS, WEAR IT"
>120 PRINT "WHERE THERE'S SMOKE, THERE'S
 FIRE"
>130 PRINT "LOOK BEFORE YOU LEAP"
>140 PRINT "BREVITY IS THE SOUL OF WIT"
>150 END
>110
>130
>LIST
  100 PRINT "VIRTUE IS ITS OWN REWARD"

  120 PRINT "WHERE THERE'S SMOKE, THER
      E'S FIRE"
  140 PRINT "BREVITY IS THE SOUL OF WI
      T"
  150 END
```

您也可以使用**DEL** 指令將整個程式片斷（**PROGRAM BLOCK**）消除；例如：

```
>DEL 110,140

>LIST
  100 PRINT "VIRTUE IS ITS OWN REWARD"

  150 END
```

DEL 110，140這個命令將行號110開始到140 爲止的程式行全部消除；即使第110行並不存在，所有在110及140之間的程式行仍然被消除掉了。

增加程式行

您可以按照任何順序或在任何時間打入新的程式行，它們的行號將決定它們在程式中的位置；APPLE II 將自動地把它們加入已經存在記憶體中的程式內；您可以試著將第120行號及110行號的程式行加入上面的例子中。

```
>120 PRINT "WHERE THERE'S SMOKE, THERE'S
 FIRE"
>110 PRINT "IF THE SHOE FITS, WEAR IT"
>LIST
  100 PRINT "VIRTUE IS ITS OWN REWARD"

  110 PRINT "IF THE SHOE FITS, WEAR IT
      "
  120 PRINT "WHERE THERE'S SMOKE, THER
      E'S FIRE"
  150 END
```

改變程式行

改變一個程式行最簡單的方法就是重新輸入一次，但是有幾個原因使得這個方法並不能令人滿意，因爲重新打入是一件費時且容易出錯的工作；幸運的是有一種可以修改您已經輸入的程式行的方法，這項功能在整數**BASIC**或**APPLESOFT**中都可以經由顯現在螢光幕上的程

式行來做處理，您可以修改任何顯現在螢光幕上的資料；您可以使用 ESC 鍵與其他的幾個鍵，使得可以在螢光幕上任意移動，您可以將游標移到螢幕上所顯示的任何一個程式行的起始處，然後使用→鍵移過不需要修改的部份，同時您可以任意的替換、插入或消除這一行中的任何字。

下面就是這個方法的介紹。首先，您使用LIST指令將您想要修改的程式行顯示在螢光幕上，您也許注意到了LIST指令在顯示程式行時多佔了額外的位置，這些多餘的空間可能在編輯長的程式行時發生困難，為防止LIST指令佔用額外的空間，您使用Esc－@ 命令將畫面清除，然後打入以下神秘的指令：

```
POKE 33,33
```

為了要消除這額外的空間，上面這個命令將螢光幕的顯示寬度由 40 字減到 33 字；我們將在第四章時對 POKE 敍述作較詳細的說明。您可以使用以下的命令使螢光幕回返到每行 40 字的寬度：

```
POKE 33,40
```

游標的移動

要想在螢光幕上移動游標，您必須依順序按兩個鍵；首先按下Esc鍵，再按A、B、C或D鍵。在您想要往右、左、下或上移動游標一個位置時，您必須每次先按下 Esc 鍵，接著再按A、B、C或D鍵。圖3－1顯示這四種按鍵的順序及游標移動的情形。

如果您的機器擁有自動監督系統，您也可以使用 Esc 鍵與I、J、K、M鍵合用來移動游標，由於這四個鍵在鍵盤上的位置特殊（如同圖 3－2 所示），它們形成了一個直接的控制鍵盤。

在 Esc 鍵與I、J、K、M鍵合用時有一個重要的不同點，那就是當您按下 Esc 鍵時，APPLE II 就進入了編輯狀態（EDIT

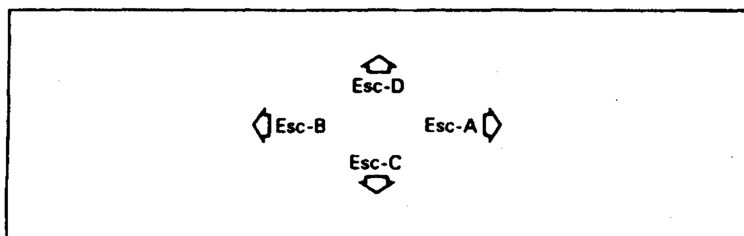


圖 3－1　游標的移動（順序的使用兩個鍵）

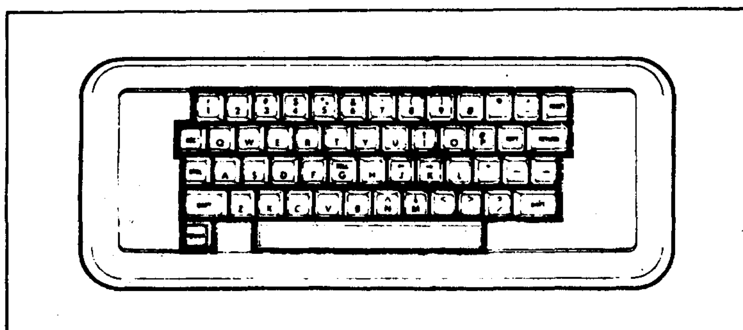


圖 3－2　游標的移動（具有自動啓動監督系統）

MODE ），這時您可以連續的使用 I 、J 、K 、M鍵移動游標，而不必每次都先按 Esc 鍵；只要您只按I 、J 、K 、M及 Rept 鍵，APPLE II 將仍然停留在編輯狀況，這使得您不必每次去按ESC鍵而可以自由地移動游標；若您想要離開編輯狀態，您只需按任何鍵（除了I 、J 、K 、M 、 Rept 、 Ctrl 及 Shift 鍵之外）就可以了。

您可以使用 Rept 鍵，在編輯狀況時快速地移動游標。

改錯〝字〞

用一個字去代替另一個字只是簡單地直接代替就可以了；您只需將游標移到想要修改的字的位置，然後打入您想要輸入的字，然後將游標移到結束的地方就可以了。例如以下的例子：

```
100 PRINT "ESTIMATED TIME OF ARRIVAL"
```

這時您可以打入 DEPARTURE ，然後您就得到以下的結果：

```
100 PRINT "ESTIMATED TIME OF DEPARTURE"
```

您這時按下 RETURN 鍵，就可以完成修改的程序了。

消除〝字〞

您可以藉著打入空白（BLANK SPACES）消除個別的字。記著在BASIC語言中，除非是在引號之間，空白並不影響任何敍述，當然您也可以使用 Esc 及A鍵（在編輯狀態時的K鍵）將游標往前移動；Esc -A 及 Esc -K並不像→鍵一般將游標所經過的字輸入記憶體內；如果您想要消除的字位於引號內，那麼您可以簡單地使用 Esc -A 或編輯狀態下的K鍵跳過不想要的字。

在螢幕上的一行中想要消除游標以後該行所有的資料，您可以使用 Esc 鍵，再用E鍵來達成這個目的，這個方法就如同您重複地按空白鍵直到行尾為止一段，不過在同一程式行但在下一螢幕行的資料並不消失；例如，如果您按下 Esc 鍵再按E鍵，如同下面例子所示的情形：

```
100 PRINT "IT IS BETTER TO HAVE LOVED AN
D LOST THAN NEVER TO HAVE LOVED AT ALL"
```

下面是發生的情形：

```
100 PRINT "IT IS BETTER TO HAVE
D LOST THAN NEVER TO HAVE LOVED AT ALL"
```

如果您這時按下 RETURN 鍵，那麼行號100的程式行將就只剩下游標以前的資料了。這時您必須使用→鍵將該行剩下的字輸入記憶體內才能按下 RETURN 鍵。

您也可以使用 ESC 鍵，再按F鍵，清除游標以後螢光幕上所有的資料。

插入"字"

在程式行中加入字是一件簡單的事；由於最後的結果並不是顯而易見的，起初可能有些使人迷惑。APPLE II 並不能將存在的字推開而空出空位讓其他的字插入，圖3－3畫出了插字的方式。使用 ESC 鍵及其他鍵使游標移到正確的位置，再將資料打入；有一件事您必須記得的是螢幕上所顯示的資料並非全部都被記憶在記憶體內。

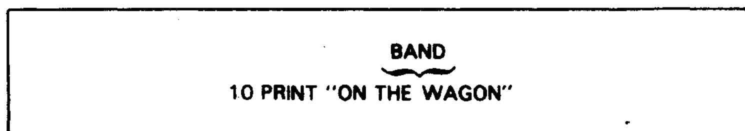


圖3－3　插字

現在我們做一個 字的示範，這個例子雖然使用的是整數BASIC語言系統，但在APPLESOFT 也是行得通的。考慮以下的程式行：

```
>NEW

>10 PRINT "ON THE WAGON"
>
```

在 WAGON 前加入 BAND 字，首先列出程式：

```
>LIST
   10 PRINT "ON THE WAGON"

>
```

只使用能使游標移動的鍵（例如 Esc 等鍵），將游標移到行號的第一個數字上，如下所示：

```
>LIST
   10 PRINT "ON THE WAGON"

>
```

這時使用→鍵將游標移到W字的上面：

```
>LIST
   10 PRINT "ON THE WAGON"

>
```

這時依順序使用 Esc 鍵與D鍵，將游標往上移動一行（這時若在游標右方仍然有字存在，您可以按下Esc 鍵，再按E鍵將它們清除）。

```
>LIST
   10 PRINT "ON THE WAGON"

>
```

打入 BAND 字：

```
>LIST                     BAND
   10 PRINT "ON THE WAGON"
>
```

僅僅使用能使游標移動的鍵，將游標移到該行的第一個位置，但千萬不要使用←鍵，因爲雖然←鍵看起來與 Esc - B 或 Esc - J 相同，但它却將所經過的字消除掉了，它將使您的插入工作報廢。

```
>LIST                     BAND
   10 PRINT "ON THE WAGON"
>
```

按下 Esc - C 鍵使游標回到原先的程式行。

```
>LIST                    BAND
   10 PRINT "ON THE WAGON"
>
```

最後使用→鍵經過這程式行的餘下部份，然後按下 RETURN 鍵，再利用LIST命令將這一程式行列出。

```
>LIST                    BAND
   10 PRINT "ON THE WAGON"

>LIST 10
   10 PRINT "ON THE BANDWAGON
```

在附錄B中的表上詳錄了編輯命令的使用方法及內容。

立即式程式行的再執行

只要是仍然可以在螢光幕上被看到的立即式程式行都可以被重新執行；您可以按照它原先的情形使它被執行，或是經過您的修改後再執行。

在以上的任何一種情形時，第一件事就是先將游標移到該程式行的起始處；使用 ESC 鍵與A、B、C、D鍵配合（若您擁有自動監督系統，那麼您就可以使用 ESC 鍵與I、J、K、M鍵了）將游標移到正確的位置，然後使用→鍵通過立即式的程式行，當然您也可以使用前面描述的編輯技巧做代換、消除、插入等等字的編輯工作。

我們藉著下面的一個計算房間體積的立即式程式行做一個示範，該房間體積爲 10 × 25 × 8 立方呎：

```
>PRINT "CU. FT. OF SPACE = ";10*25*8
CU. FT. OF SPACE = 2000
```

您可以改變這個程式行用來計算不同體積的房間；若您想將這房間的體積改爲 10 × 25 × 14，那麼首先您先將游標移到該程式行的起始位置，然後使用→鍵與 REPT 鍵將游標往右快速移動，當游標移到 8 的位置時，將兩個鍵同時放鬆，若您移動的過了頭或者尚未到達適當位置時

就已經將鍵鬆掉了，那麼您可以使用←鍵或→鍵將游標移到正確的位置；當然您若怕發生過頭或不及的情形，您也可以連續按→鍵 33 次使游標達到 8 的位置。

當游標位於 8 的位置上時，您就可以打入新房間的行列值 14 了，接著按下 RETURN 鍵。

```
>PRINT "CU. FT. OF SPACE = ";10*25*14
CU. FT. OF SPACE = 3500
```

程式語言

我們藉著程式語言（PROGRAMMING LANGUAGE）與電腦作各式各樣的聯繫；程式語言有許多不同的種類，例如：BASIC 語言為一般使用的語言，而有些語言却是為了發展特殊的功能而產生的；比如說，為了商業、科學研究、繪圖、文字處理或者其他的用途等等。程式語言就像方言一般地有很大差別，除了BASIC語言外，一般常被使用的語言還有PASCAL、C、FORTRAN、COBOL、APL、PL/M、PL-I及FORTH等。

APPLE II除了BASIC及PASCAL外尚能使用許多的程式語言，但這本書將專注於描述APPLE II的BASIC語言。

不論何種程式語言，它們都有一定的規則必須遵循，這些規則一般被稱為語法（SYNTAX），每一種的程式語言都有它特定的語法。

程式語言與我們所說的語言一樣都有它的方言（DIALECTS），方言使得程式語言在語法上有一些改變，例如APPLE II 就具有兩種不同語法的方言：整數BASIC及 APPLESOFT；由於兩者語法的不同，因此並不能夠互相通用；更有甚者，用APPLE II BASIC語言寫的程式未必能在其他的電腦上執行。然而如果您已經學會了APPLE II 的BASIC語言，那麼在您學習其他機器的BASIC語言時就不會那麼困難了。

有些程式語言的語法是顯而易見的，例如，在本章開始時的加法及減法的語法，即使您並非一位程式設計師，您也可以很快地瞭解它們的意思，但有些語法却是完全獨一的而且似乎是無意義的，您最好不要想用推理的方式去瞭解語法，因爲它們一般都是無理可循的。例如：爲何要使用“ * ”來代表乘法呢？正常的情況，您必定是使用“ × ”來表示乘法，但是幾乎所有的程式語言都是使用“ * ”來代表乘法，而用“ / ”代表除法，這並沒有什麼特別的原因，只是因爲鍵盤上並沒有“ ÷ ”號，故而必須選擇一個符號來代替它罷了！

BASIC語言的組成元素

許多的BASIC語言的語法考慮到的都是個別的敍述，語法分別與三個主要的組成部份發生關連：行號、指令及資料，我們現在逐一地描述於下。也有一些規則使得整個程式能夠維持成一體，例如，敍述的順序；在這一章中我們也將在適當的地方做解說。

行號的規定

我們已經在前面討論過行號的問題，在重新複習以後我們將再詳細的敍述一些較深入的規則。在間接式的情況下，每一個BASIC程式行都必須有一個獨一的行號，行號決定程式行執行的先後順序，行號最小的程式行最先執行，而最後執行的程式行必定具有最大的行號。

整數BASIC語言准許一位到五位的整數行號，它們的範圍是在∅與32767之間。

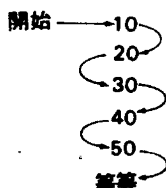
APPLESOFT同樣准許一到五位的整數行號，但是範圍却擴大到∅與63999之間。

行號當做位置

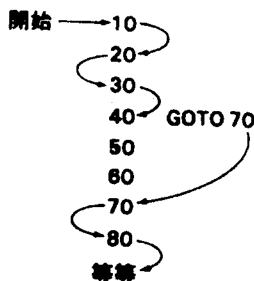
行號可以當做定程式行位置（ADDRESS）的一種方法；這是一個重要的概念，因爲每個程式都包含兩種型式的敍述：

(1)修改或建立資料的敍述。
(2)控制執行順序的敍述。

指令執行的先後必須依照一定的順序完成，可說是一種簡單的概念；一般來說，程式的執行必定從第一個敍述開始，然後按順序執行，就如同下面所闡述的情形：



但是我們可以很快的發現，事實上許多的程式執行常常並非按照順序來完成的，這使得行號變得非常重要，因爲您可以使用行號來代表執行順序的改變，這種情形可以由以下的圖形做一個闡述：



空　白

一般說來，您可以使用多餘的空白位置使您的程式更容易閱讀，但是由於空白同樣的佔用記憶體位置，所以APPLE II 會自動地取消程式行中的空白（在您按下 RETURN 鍵之後），但當您使用 LIST 指令顯示程式時，APPLE II 將會將空白加入程式行中，使得您的程式易於閱讀；回憶一下，您應該記得可以使用 TEXT 指令或打入 POKE 33,33消除多餘的空白。當然您可以用 POKE 33,40 來恢復正常的情況。

在引號之間使用空白時您必須小心從事；例如，您可以比較以下的兩個指令：

```
PRINT "ENTER INVOICE DATE"
ENTER INVOICE DATE

PR INT "      ENTER INVOICE D ATE"
      ENTER INVOICE D ATE
```

資　料

電腦程式的最重要工作就是輸入（INPUT）、處理（MANIPULATE）及輸出（OUTPUT）資料（DATA），所以一個程式語言處理資料（不論是文字資料（TEXT）或是數字資料（NUMBER））的能力，確實是非常重要的。現在我們將描述您在APPLE II BASIC語言中可能遇到的資料型式。

字　串

"字串"（STRING）就是包含在兩個引號間的字或一連串的字，我們已經在PRINT指令中使用了字串，使它們顯示在螢光幕上。以

下是幾個字串的例子：

"IGNORANCE IS BLISS"
"ACCOUNT 4019-181-324-837"
"NICK CHARLES"
"SAM & ELLA CAFE"
"MARCH 18, 1956"

除了少數例外的情形，字串可以包含任何鍵盤上所有的字鍵或數字鍵；這些例外的鍵是←鍵、→鍵、 RETURN 鍵、ESC 鍵、CTRL -H、CTRL -M、CTRL -U及CTRL -X ，它們都使得游標在螢幕上移動或結束您正在工作的程式行。

字串的長度可以由 ∅ 到255 個字，一個沒有任何字的字串被稱為虛字串（NULL STRING）。

您可以按某些字鍵的組合而得到一些看不見的字，例如，您使用 CTRL - G可以使得APPLE II 發出"嗶"的聲音，您可以將CTRL -G 放在字串內，然後將它們使用在PRINT 指令裏。

PRINT "" 在引號之間按 n 次 CTRL-G

在這個例子中您雖然看不到這個字，但您可以聽到它的聲音；有些字却是既看不到也聽不到的，它們是用來控制印表機的功能、訊號交換設備（COMMUNICATIONS DEVICES ）以及其他您能加在APPLE II 上的設備。

在附錄 I 中有完整的APPLE II 的字集（CHARACTER SET），同時告訴您那一個鍵可以產生那一個字。

在 APPLESOFT 中使用字串時，您可以使用 CHR$ 這個功能產生一些不能用按鍵產生的字，這種情形我們將在第四章時做一個描述。

數 目

兩種型式的數目（NUMBERS ）可以被儲存在APPLE II 內：整數（ INTEGERS ），不包含任何小數部位；實數（REAL NUM

BERS），可以有小數部位；如您所猜測的，整數BASIC語言只認識整數，而APPLESOFT則不論整數或實數都能處理。

您在使用數目時，不可以使用逗號“，”；例如，您必須使用32000而不是32,000。

整　數

整數就是一個不包含小數部份或小數點的數目，可以是正數或負數，但如果不包含正負號則這個數目就被認爲是正整數。最小的整數是－32767，最大的則是32767。下面是整數的一些例子：

0
1
44
32699
−15

實　數

實數可以是一個整數、整數及帶有小數部位或僅僅有小數部位的數目；可以是正數或負數，如果不包含正負號，那麼這個數目就被認爲是正數。最小的實數是：

−1000000000000000000000000000000000000000

最大的實數則是：

1000000000000000000000000000000000000000

以下是一些實數的例子：

5
−15
65000
161
0
0.5
0.0165432
−0.0000009
1.6
24.0055
−64.2

當一個小數值較 .000000000000000000000000000000000000003. 還要接近 ∅ 時，這個數目就被認爲是 ∅ 了。

科學表示

很大的或很小的數目在APPLESOFT 中就用科學表示（SCIENTIFIC NOTATION）的方式表現出來；任何在小數點之前多過九位數字的數目及任何小數部位較±0.1 更接近 ∅ 的數目都將使用科學表示的方法表現出來。

科學表示的數目具有以下的格式：

NUMBER E±ee

NUMBER ：是一個整數、小數或整數帶小數的數目。NUMBER包含著這個數目的有效數字 —— 稱爲係數（COEFFICIENT ）。如果這個數目沒有小數點，那麼機器自動假設小數點位於係數的右邊。

E ：E字代表指數（EXPONENT ），永不改變。

± ：±是可以省略的正負號。

ee ：可爲一位或兩位的指數值，指數值指出這個數目的眞實值；也就是說，如果是正的指數值，

那麼小數點必須往右移動而得到正確的數值；反之，則向左移動，表示正確的數值。

以下是幾個科學表示數值的例子：

標準表示	科學表示
1000000000	1E+09
.000000001	1E-09
200	2E+02
-123456789	-1.23456789E+09
-.00000123456789	-1.23456789E-06

如同您所見到的，科學表示提供了表示過大或過小數目一個便捷的方法。實數的最大值及最小值分別是 1 E + 38 及 -1 E + 38 ；同樣地，最接近 ∅ 的數目可以是 3 E - 38 。

四捨五入

我們在前面提過實數最多可以有九位有效數字；對一個大於 1 或小於 - 1 的數目而言，那只有最左邊的九位數字可以不是 ∅ ，APPLE Ⅱ 將超過九位數字以後的數字四捨五入。以下是幾個例子（太大的數目用科學表示方法表現出來）：

```
]PRINT 1234567891
1.23456789E+09

]?-123456789123456789
-1.23456789E+17

]?-150000475.75
-150000476

]?90000000.7558
90000000.8
```

小數（在 1 與 - 1 之間的數目）同樣地受到以上的限制，在這種情形下，這九位有效數字從小數點的右邊第一位開始算起。以下是幾個例子：

```
]PRINT .1234567891
.123456789

]?-123456789123456789
-1.23456789E+17

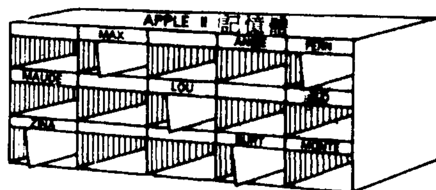
]?-123456789 123456789
-1.23456789E+17

]?.00000000009000000007558
9.00000008E-10
```

變 數

到目前爲止我們討論的資料僅限於常數（CONSTANT VALUES）；有時候使用一個名字來代表資料較直接使用它的值來得方便得多，這也就是變數（VARIABLES）的功用。

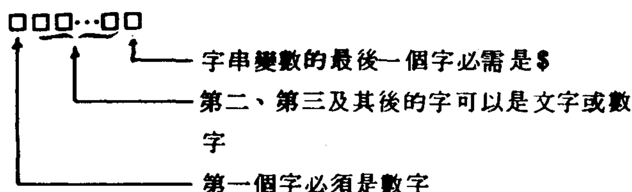
如果您研習過基本代數，那麼您對變數及變數名稱的概念應該沒有任何麻煩；如果您從來沒有接觸過代數，那麼您可以想像變數名像一個郵箱，任何放在郵箱內的東西，就由這個郵箱的名稱來代表，直到有新的東西放入這個郵箱內爲止；在電腦的術語中，我們說一個值被存在（STORED）一個變數內。



一個變數的眞正功能就在於它不必永遠代表某一個值，它可以代表任何合法的值，您可以在程式執行時隨意地改變它；BASIC語言中有許多的敍述可以達成這個目的，我們將在後面討論。

整數BASIC語言的變數名稱

在整數BASIC內變數名稱的長度可以從1到100個字；下面是它的一般規則：



因此在變數名稱的最後一個字告訴整數BASIC語言那一種資料是它所代表的——字串或數字。

字串變數（STRING VARIABLES）可以代表由∅到255個字長度的字串，空白位置照算一個位置。在您使用字串變數之前，您必須指出該字串可能的最大長度，您可以使用DIM指令達成這項指定（這個指令將在稍後描述），如果您指定的不正確，那麼您將得到*** STR OVFL ERR的訊息。下面是幾個合法及不合法字串變數的例子：

合 法	不合法
Legal	**Illegal**
A$	$
CUSTNAME$	9$
PART1$	BRAND.NAME$
RESPONSE$	
X8$	

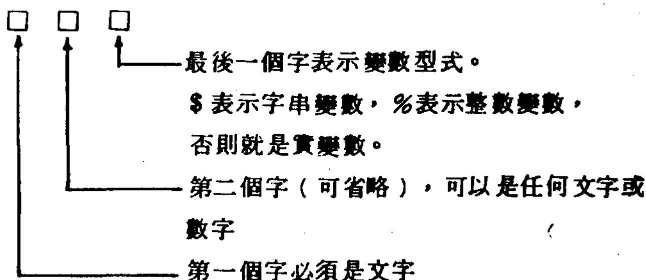
數字變數（NUMERIC VARIABLE）在整數BASIC內所代表的數目必在－32767及＋32767之間，如果數值超出了這個界限，您將得到***>32767 ERR的錯誤訊息　以下是一些整數BASIC中的

合法及不合法數字變數的例子：

合　法	不合法
Legal	Illegal
A	APPLICANT'S AGE
CUSTZIPCODE	3X4Z
XO	$TOTAL

APPLESOFT中的變數名稱

在 APPLESOFT 中變數名稱可以是一個、兩個或三個字的長度，下面是它的規則：



因此在 APPLESOFT 中變數的最後一個字也表示出這個變數所代表的資料型式。

APPLESOFT 中的字串變數可以代表由 ∅ 到255 個字長度的字串，您不必如同整數BASIC中一般的指出這個字串的長度。以下是一些合法及不合法字串變數的例子：

合　法	不合法
A$	O$
MN$	MI$
F6$	77$

整數變數只能代表在－32767 及＋32767 之間的數目，如果您超過了這個極限，您將得到？ ILLEGAL QUANTITY ERROR 的錯

誤訊息；如果您想要將實數存入整數變數內，APPLESOFT 將會自動地將實數轉換成整數，我們將簡短的介紹轉換的規則。以下是一些APPLESOFT中合法及不合法的整數變數：

合　法	不合法
A%	A$%
B%	31%
A1%	3D%
X4%	

實數變數一般被限制在 -10^{38} 到 $+10^{38}$ 之間，但有時您也可以計算到 $\pm 1.7 \times 10^{38}$。如果您想要存一個太大的數值到實數變數內，您將得到？OVERFLOW ERROR 的錯誤訊息。

當一個實數變數的值比 $\pm 2.9388 \times 10^{-39}$ 更接近 ∅ 時，APPLESOFT 將把它轉換成 ∅。

需要記住的是，實數變數可以有整數值，因爲一個整數就是一個小數部份是 ∅ 的實數。以下是幾個合法及不合法的實數變數的例子：

合　法	不合法
A	0
B	7B
A1	A#
AA	
Z5	

APPLESOFT中的長變數名稱

變數名稱的長度可以超過兩位字（也可在最後一位加上％或$，分別代表整數或字串變數），但在 APPLESOFT 中只有最初的兩個字或數字被認爲有效；因此PRICE 1 及 PRICE 2 代表同樣的變數名稱（因爲它們同樣以 PR 開始）。但是 PRICE 1 及 PRICE 1 %却是不同的，因爲它們的最後一位代表了不同型式的變數。

APPLESOFT允許變數名稱的長度到 238 個字。

下面是一些超過兩位字長度的合法及不合法的變數名稱：

合 法	不合法
COUNTER%	ITEM#%
ACCOUNTBALANCE	2NDRATE
NAME$	CUSTOMER. ADDRESS$

如果您使用的變數名稱超過兩個字，那麼您必須記住以下幾點：

㈠只有前面兩位字及代表變數型式的符號（ $ 或% ）被認爲有效。不要使用延長的名稱如同LOOP 1 %及LOOP 2%，因爲它們代表相同的變數：LO% 。

㈡多餘的字佔用多餘的記憶體位置，往往這使得您的程式所能使用的記憶體位置變小。使用較長變數名稱的好處在於使您的程式易於閱讀；例如，PARTNO就較PA在庫存程式中代表貨號顯得較有意義。

保留字

所有定義BASIC語言中敍述動作的字就被稱爲保留字（RESERVED WORDS ）。在附錄F中詳列了整數BASIC及APPLESOFT 中的保留字，您可以在本章中遇到許多的保留字，其他的將在本書的其他章節做描述。

當一個BASIC程式被執行時，APPLE Ⅱ將掃描每一個BASIC的敍述，尋找出組成保留字的字串，當然，包含在引號間的字是一個例外。所以如果您使用的變數名稱中隱藏了保留字，那麼就會產生麻煩了，因爲APPLE Ⅱ並不能夠在BASIC中由變數名稱的位置來認明它，因此您必須特別小心，不要在變數名稱中使用保留字；這對短的保留字而言，特別要注意，因爲它們很容易地就會被誤用。

矩 陣

矩陣（ARRAYS）就是一個自動命名一大堆變數名稱的工具，它經常被使用在許多不同型式的電腦程式內；如果您對矩陣並不熟悉，那麼您必須詳細研讀下面的章節，因爲它們包含了許多您在撰寫程式時需要注意的事。

矩陣並不是很複雜的概念，當您擁有兩個或更多的資料，您可以給這些資料一個變數名稱，而不是給每一個資料一個變數名稱，這時這個包含許多資料的變數就被稱爲矩陣，它的名字就是矩陣名稱，它所包含的個別資料就被稱爲矩陣元素（ARRAY ELEMENTS）；矩陣元素都由一個號碼來代表，當您想要使用它時，只需要指出它所在位置的號碼即可，這個號碼就被稱爲脚註（INDEX）。

矩陣是一種用來描述相關資料的有效方法；您想想看，假設有一個包含200個數目的表，您是喜歡給每一個數目一個名稱呢？還是給整個表一個名稱再用每個數目的位置做一個脚註呢？當然後者是較前者簡單得多了，而這也就是矩陣的最大功用。

現在舉一個矩陣的例子，假設一個擁有10個房間的小旅社，如果想要知道每間房裏住了什麼人，那麼您可能每個房間都給它一個名稱，然後與住在裏面的人相對應。

JONES	SMITH	DOE		LITTKE	ALTON	DAVIS	HANSON	SHORTEN	
R1$	R2$	R3$	R4$	R5$	R6$	R7$	R8$	R9$	R10$

或者您可以把所有的房客都放在矩陣裏。

JONES	SMITH	DOE		LITTKE	ALTON	DAVIS	HANSON	SHORTEN	
R$(1)	R$(2)	R$(3)	R$(4)	R$(5)	R$(6)	R$(7)	R$(8)	R$(9)	R$(10)

在這個例子中，R$就是一個矩陣的名稱，它包含有10個元素，每個元素就代表它所對應房客的名字。包含在括弧內的脚註跟隨在變數名稱的後面，使得某個房間內房客的名字就由這個變數名稱及脚註所代表；例如，第三個房間的房客的名字就放在R$(3)裏面，也就是DOE。

上面這個矩陣的例子用的是字串矩陣（STRING ARRAYS）

因爲字串對大多數的人而言較易瞭解。實際上，整數BASIC語言只允許數值矩陣（NUMERIC ARRAYS）。

在APPLESOFT中，矩陣可以代表整數變數、實數變數或字串變數；但是，一個矩陣變數只能代表一種資料型式；換句話說，也就是一個變數不能同時將整數與實數混合使用，除非一個實數變數，它可以有整數值，但是整數變數就不可以有實數值。每種型式的矩陣所佔用的記憶體位置也不相同，在附錄G中有詳細的資料。

矩陣行列

當在整數BASIC中使用矩陣時，您必須告訴APPLE Ⅱ這個矩陣元素的數目，您可以使用DIM指令達成這個目的；我們將在本章稍後的地方講解這個指令。

APPLESOFT中的矩陣，如果它的元素數目不超過10個時，您可以不必告訴電腦這個矩陣的元素數目。

矩陣在APPLESOFT中可以有不只一個的行列，也就是說可以用多重的脚註選擇一個元素。矩陣只具有一個行列就如同只具有一行數目的表一般，一個脚註代表這一行中的一個數目；矩陣具有兩個行列時，就可以表示二度空間的表，一個脚註表示元素在行（COLUMN）的位置，另一個脚註則代表列（ROW）的位置；您可以使用三個脚註來代表三重的資料位置，就如同代表一個立方體的三重座標一般；對於四重或更多的行列是比較難用想像、比較方式瞭解的，但在數學上來說它們並不比一個小的行列來得複雜。

我們可以將上面例子中的房客數目擴充而用二階的行列式來表示；試想一個有八層樓而每層有十個房間的旅社，以下有四種方法將八十名客人記錄下來，並能簡單地發現人名與房號的對應；第一：我們可以將每個房間設定它獨一的變數名稱；第二：整個旅舍可以用一個矩陣來表示；第三：每一層樓可以用一個包含十個元素的矩陣表示；第四：我們可以用一個二階的矩陣代表整個旅舍；第四種的選擇我們可以用以下的

表來表示：

H$(8,1)	H$(8,2)	H$(8,3)	H$(8,4)	H$(8,5)	H$(8,6)	H$(8,7)	H$(8,8)	H$(8,9)	H$(8,10)
H$(7,1)	H$(7,2)	H$(7,3)	H$(7,4)	H$(7,5)	H$(7,6)	H$(7,7)	H$(7,8)	H$(7,9)	H$(7,10)
H$(6,1)	H$(6,2)	H$(6,3)	H$(6,4)	H$(6,5)	H$(6,6)	H$(6,7)	H$(6,8)	H$(6,9)	H$(6,10)
H$(5,1)	H$(5,2)	H$(5,3)	H$(5,4)	H$(5,5)	H$(5,6)	H$(5,7)	H$(5,8)	H$(5,9)	H$(5,10)
H$(4,1)	H$(4,2)	H$(4,3)	H$(4,4)	H$(4,5)	H$(4,6)	H$(4,7)	H$(4,8)	H$(4,9)	H$(4,10)
H$(3,1)	H$(3,2)	H$(3,3)	H$(3,4)	H$(3,5)	H$(3,6)	H$(3,7)	H$(3,8)	H$(3,9)	H$(3,10)
H$(2,1)	H$(2,2)	H$(2,3)	H$(2,4)	H$(2,5)	H$(2,6)	H$(2,7)	H$(2,8)	H$(2,9)	H$(2,10)
H$(1,1)	H$(1,2)	H$(1,3)	H$(1,4)	H$(1,5)	H$(1,6)	H$(1,7)	H$(1,8)	H$(1,9)	H$(1,10)

如您所見到的，在這個二階矩陣中第一個脚註表示的是該旅舍的層數，而第二個脚註表示的就是該層中的房間號碼了；所以在R$（3，2）中所存放的就是住在三樓第二個房間內的旅客名字。

APPLESOFT中的矩陣階數可以擴展到88階，而每階所擁有元素的數目並沒有特別的限制；當然由於每個元素需要佔用記憶體的位置，所以記憶體的大小就限制了元素的數目了。

運算式

在這節中我們將告訴您如何用運算式（EXPRESSIONS）將數值與變數結合在一起；我們曾經在前面使用過運算式計算立即式命令中的簡單算術運算，回想以下的指令：

```
PRINT 4+6
10
```

這個指令告訴APPLE Ⅱ將4及6相加，並將它們的和顯示出來；以下的這個指令

```
PRINT A+B
0
```

告訴APPLE Ⅱ將數值變數A和B相加，並將它們的和印出。

加號（＋）表示加法，標準的電腦術語中將加號認作一個運算（OPERATOR），由於加號是一個算術的動作——它表示加法，所以加號就是一個算術運算（ARITHMETIC OPERATOR）。

算術運算是最容易瞭解的了，在我們小的時候，我們都學過加法、減法、乘法及除法；當然除此之外，還有許多其他種類的運算：字串運算（STRING OPERATORS）、相關運算（RELATIONAL OPERATORS）以及布爾運算（BOOLEAN OPERATORS），它們也是很容易就能使您瞭解的，只是必須多加一些解釋罷了。

每一種的運算都定義了一種型式的運算式，它們分別是：算術運算式（ARITHMETIC EXPRESSIONS）、字串運算式（STRING EXPRESSIONS）、相關運算式（RELATIONAL EXPRESSIONS）以及布爾運算式（BOOLEAN EXPRESSIONS）。

運算式中運算的優先執行順序

運算式中可以同時擁有數個運算。例如，以下的例子：

```
PRINT A+B/10
0
```

在同一個運算式中使用了加法及除法；這裏有一個標準的規則決定在一個運算式中各個運算的先後順序，我們將告訴您在每一種型式的運算式中先後順序（PRECEDENCE）的規則，依序是字串的連接（STRING CONCATENATION），接著是整數、實數、相關、布爾及混合型式的運算式。首先讓我們看看超越標準先後順序規則的方法。

超越先後順序的方法

您可以藉著使用括弧的方法改變APPLE Ⅱ執行運算的先後順序，任何包含在括弧內的動作都是被優先執行的；當一個運算式擁有多重

的括弧，那麼APPLE Ⅱ將從左到右依序執行它們。

當一組括弧包含在另一組括弧內時，我們就稱這種情形為多重結合（NESTING），在這種情形下，APPLE Ⅱ將自內而外地依序計算它們；括弧可以有任何數目的層數，您可以自由的使用它們使得您的計算顯得清楚而條理分明。

以下就是幾個立即式情況下使用括弧的情形：

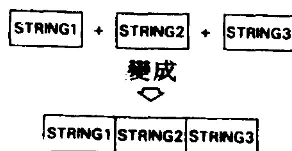
```
PRINT (2+10)*3
36

>
PRINT ((2+10)*3+31)*10
670

>
PRINT -(2^(3+8/4))
-32
```

字串的連接

您可以將許多的字串接連在一起形成一個較長的字串，這就稱為連接（CONCATENATION），您可以在下面的圖中看得清清楚楚：



使用連接的方式，您可以將字串加長到255個字的長度。

在整數BASIC中並不存在字串連接的運算，所以您必須使用本章末尾處描述的方式來做連接的工作（如果在此處就做說明，似乎稍嫌早了一些）。

APPLESOFT 中使用加號（＋）做為連接運算；以下就是一些APPLESOFT中字串連接的例子。

"OVER" + "DUE"	變　成	"OVER DUE"
"MONTHLY" + " " + "REPORT"	變　成	"MONTHLY REPORT"
"WEEKLY" + R$	變　成	WEEKLY後面接著R$的值
A1$ + YA$ + C$(1)	變　成	A1$值後接YA$再接C$(1)值

整數運算式

整數運算式是一個算術運算式，其中僅祇包含了整數變數及整數常數。我們將在"混合型式的運算式"這一節中描述包含整數及實數的運算式。

整數運算式中的運算包括了加號（＋）、減號（－）、乘號（＊）、除號（／）及指數符號（∧），您也可以使用放在最前面的負號（－）表示負數值。運算的先後順序依下而定：負號第一優先，接著是指數部份，乘法及除法次之，最後則是加法及減法，同樣優先順序的運算先後，則依自左而右的規則。

下面就是一些在整數**BASIC**中整數運算式的例子：

100 － 30 ＊ 2	結果是	40
－9 ∧ 2	結果是	81
A/B＊C	結果是	A的值被B的值除，然後商數的整數部分與C相乘
D＋X＊3	結果是	X的值乘3，然後加上D的值
5/2＊2	結果是	4（5/2的商數被變成整數值2）

以下是一些在**APPLESOFT**中整數運算式的例子：

－120/2＋100	結果是	40
2 ∧ 3 ＊ 2	結果是	16

N1% * N2%/N3%	結果是	N1%的值與N2%的值相乘，其積再被N3%的值除
AA%/AB%/AC%	結果是	AA%的值被AB%的值除，其商再被AC%的值除
5/2 * 2	結果是	5（5/2的商數並沒有被轉換成整數）

在整數BASIC中您可以在整數運算式中多使用一個運算，它將除法中的餘數留存下來（當不能整除時），這運算就是**MOD**，它與乘法及除法有相同的優先順序。以下就是一些**MOD**的例子：

4 MOD 3	結果是	1
3 * 5 MOD 4	結果是	3
(41+2)/25MOD A	結果是	18除以A後的餘數
3 MOD 4	結果是	3

實數運算式

APPLESOFT 可以允許另外一種的算術運算式，雖然它的運算與整數運算式的相同，但它可以有實數的存在，而它的運算的執行優先順序與整數運算式的相同；依序分別是：負號、指數、乘法及除法、加法及減法。以下是一些實數運算式的例子：

87.5−4.25 * 2	結果是	79
1.5 ∧ (3/2/2)	結果是	1.35540301
AL *(PL−3.1 * CB)	結果是	PL的值減去3.1乘上CB的值，再與AL的值相乘
7.5 * 2/5	結果是	3

相關運算式

相關運算的功能在於能分辨兩個值的相關關係，您可以比較第一個值較第二個值是大於、小於、等於或是不等於、大於或等於（GREATER THAN OR EQUAL）、小於或等於（LESS THEN OR EQUAL）這幾種情況；而它們可以是常數、變數或是任何型式的運算式（在整數BASIC中有較嚴格的限制）。如果在相關運算一邊的值是一個字串，則在另一邊的值也必須是一個字串。

如果比較的結果如同運算所表示的情形，那麼這個相關運算式的結果就是數值 1；相反的，若比較的結果與運算元所表示的並不相符合，則它的結果就是 ∅。

除了如同表 3－1 所示唯一的例外情況之外，整數BASIC與APPLESOFT 的相關運算都是相同的。

表 3－1 相關運算

整數BASIC 運算	作業	APPLESOFT 作業
<	Less Than*	<
>	Greater Than*	>
=	Equal To	=
# or < >	Not Equal To	< > or > <
>=	Greater Than or Equal To*	> = or = >
<=	Less Than or Equal To*	< = or = <

* 在整數 BASIC 的字串中不可使用

所有的相關運算都有相同的優先順序，所以它們運算的先後順序依照從左至右依序執行。

以下是一些相關運算式的例子：

1 = 5 − 4	結果是　1（眞）
14 > 66	結果是　ϕ（假）
15 >= 15	結果是　1（眞）
"AA" > "AA"	結果是　ϕ（假）
"ANDERSON" < "ASHLEY"	結果是　1（眞）
(A=B)=(A$ > B$)	結果依照變數的值而定，如果A的值與B的值相等，而A$較B$的值大，則這個運算式的結果就是1。

如果只有相關運算的觀念，那是非常容易瞭解的；但是在BASIC中ϕ與1分別表示假與眞，但也可以被用在實數或整數的運算式中，由於它們是完全絕對的，故而並非十分容易瞭解；例如：（1＝1）＊4這個運算式具有什麼意義？如果不在BASIC語言內，那麼上面這個運算式不具有任何意義，但在BASIC語言中，（1＝1）就表示眞，而眞也就是數值1，因此上面這個運算式就等於1＊4，而它的結果就是4；您可以在BASIC語言內使用包含有相關運算式的運算。以下就是幾個例子：

25 + (14 > 66)	等於　25 + 0
(A + (1 = 5 − 4)) * (15 >= 15)	等於　(A + 1) * (1)

字串的比較

您也許在懷疑APPLE Ⅱ使用什麼樣的規則來比較字串。以下是兩種考慮：第一是字串的長度，不同長度的字串（記得空白也被算入字串的長度），就是不相等，如果一個較短的字串與一個較長字串的前半部相同，那麼這較長字串就大於那個較短的字串（這只在APPLESOFT中適用）；第二點要考慮的是相同長度的字串是否擁有相同的

字，而且依照相同的次序排列。在整數BASIC中您可以使用＝及＃或<>運算來比較兩個字串。字串的比較是由左邊第一個字開始，一個一個地與另一字串相比較——第一個字與另一字串的第一個字相比，第二個與第二個相比，如此繼續比較下去，直到其中之一的字串已經比較完畢，或是有任何兩個字的比較不能相符時則停止這個比較的程序。

APPLESOFT 比較字的大小是依它們的相關次序而定的；例如字母的大小順序分別是A＜B，B＜C，C＜D，……，而在字串中出現的數字則依 ∅＜1，1＜2，2＜3，……的次序，至於其他在字串中出現的字如＋、－、$ 等等，則依附錄 I 中的順序而定大小次序。

表3－2 布爾眞值表

AND 操作僅在兩者值都是1時才會得到1

1 AND 1 = 1	1 AND 0 = 0
0 AND 1 = 0	0 AND 0 = 0

OR 操作只要有1個值爲1就可得到1

1 OR 1 = 1	1 OR 0 = 1
1 OR 1 = 1	0 OR 0 = 0

NOT 操作與原值相反

NOT 1 = 0

NOT 0 = 1

布爾運算式

布爾運算提供了程式中做邏輯決定的能力，因此它們經常被稱爲邏輯運算（**LOGICAL OPERATORS**）。以下有四種標準的布爾運算：**AND**、**OR**、**EXCLUSIVE OR**及**NOT**。**APPLE Ⅱ** 中的**BASIC** 語言只提供了三種運算：**AND**、**OR**及**NOT**。

如果您並不瞭解布爾運算，那麼用一個簡單的例子可以使您瞭解它

。假設您在超級市場中選購兩個小孩早餐時的麥片，那麼 **AND** 布爾運算表示如果這兩個小孩都要麥片，那麼您才會買它；**OR** 布爾運算表示只要有任何一個小孩要，那麼您就會買；**NOT** 則表示相反的意思，如果B小孩不同意A小孩的意見，那麼B小孩的決定就是A小孩決定的 **NOT** 。

電腦並不能瞭解比喻的意思，它們只與數目發生關連，因此布爾運算將它的值限於1或ø（眞或假）。由於布爾運算與ø及1發生關連，因此它們常與相關運算式合用。布爾運算也可以與其他型式的運算合用，這一點我們將在下一節中敍述。

表3－2表示出布爾運算式的運算情形，這裏就被稱爲眞值表（**TRUTH TABLE**）。布爾運算具有相同的優先順序，當在同一個運算式中出現了幾個布爾運算時，它們執行的先後順序是依照由左至右而定的。以下是一些布爾運算式的例子：

NOT((3+4)>=6)	結果是	ø（假）
("AA"="AB") OR((8*2)=4△2)	結果是	1（眞）
NOT("APPLE"="ORANGE") AND(A$=B$)	結果是	1（眞）如果A$與B$相等 ø（假）如果A$不等於B$

混合型式的運算式

經常一個運算式包含著多種型式的值，這在 **APPLESOFT** 中尤其是容易發生的，因爲 **APPLESOFT** 可以包含兩個不同型式的數值——整數及實數。我們已經在討論相關運算式及布爾運算式中介紹了混合型式運算式的概念，您可以自由地混合使用任何型式的值，當然字串

是不可以在整數、實數、布爾運算式中出現的了，它只能使用在字串或是相關運算式內；以下就是一些混合型式的運算式：

合 法	不合法
3.1416 * (R ^ 2)	1600 + "PENNSYLVANIA AVENUE"
A% >= B/3	ST$ <A%
43 AND 137	A$ AND B$
1 OR 4E + 10	NOT (A$) = B$
(A$ = B$) AND -6.25	NOT(A = B) OR C$

當APPLE Ⅱ碰到混合型式的運算式時，它有幾件事必須逐一地解決；首先它要考慮的是運算優先順序的問題；表 3 — 3 對所有的運算的優先順序做了一個排列，我們由表中可以看出任何包含在括弧內的運算永遠是最優先被執行的，當有多層的括弧存在時，APPLE Ⅱ將先執行最內層括弧中的運算，依次往外推，最外層的括弧將被最後執行（您可以回憶一下我們前面所提的多重結合的概念），接著，APPLE Ⅱ將計算數學運算式，然後是相關運算式，最後才是布爾運算式。

如同我們前面所提的，相關運算式依照運算所表示的關係的眞或假而得到 1 或 ∅ 的值，因此，相關運算式是可以寫在整數運算式或實數運算式內的。

您也可以在布爾運算式中使用混合型式的運算式，因爲當布爾運算式開始執行時，任何存在該運算式中的值都會先被轉換成 ∅ 或 1 的型式；數值是依照以下的規則做轉換的：如果數值是 ∅ ，則它的值不變，至於任何不是 ∅ 的數值則被轉變成 1 。

BASIC 語言無法自動地將字串轉變成數值，所以字串只能寫在相關運算式內，至於在整數、實數及布爾運算式中都是不合法的。

在 APPLESOFT 中不論整數或實數都可以出現在實數、相關或布爾運算式中。

當整數出現在實數運算式中時，它將被暫時地轉換成實數值以便運算式的計算，而這個運算式的最後結果是實數或是整數則依照這個運算式的上下程式行而定， APPLESOFT 將自動地作適當的轉換。

實數轉變成整數時是將該實數的小數部份去掉，並取小於該實數的

表3－3　運　算

	優先順序	整數BASIC 運算	APPLESOFT BASIC 運算	意義
	高 9	( )	( )	括號表示出運算的優先次序
數學運算	8	^	^	乘冪
	7	—	—	負號
	6	*	*	乘法
	6	/	/	除法
	6	MOD	不能使用	餘數
	5	+	+	加法
	5	—	—	減法
相關運算	4	=	=	等於
	4	#	<> 或 ><	不等於
	4	<	<	小於
	4	>	>	大於
	4	<=	<= 或 =<	小於　等於
	4	>=	>= 或 =>	大於　等於
布爾運算	3	NOT	NOT	非
	2	AND	AND	且
	1 低	OR	OR	或
**	整數BASIC才可以使用			

整數，以下就是幾個例子：

1.1	變成	1
1.9	變成	1
－1.1	變成	－2
－1.9	變成	－2

BASIC語言的敍述

現在我們已經準備將部份的BASIC敍述指令爲各位做說明了，您可以用BASIC命令來從事各項的動作。將指令（COMMAND）與敍述（STATEMENT）交換著稱呼是一種習慣，同時它們經常是代表同一個意義的；嚴格地說，一個指令就是在立即式情況下的一個命令，而這同樣的命令在間接式情況下就是一個敍述。

每一個命令執行一個特別的工作，這一章將把設計程式的概念介紹給您，並將告訴您敍述的使用方法，但我們並不詳細的描述敍述，您必須詳細地研讀第八章以便瞭解敍述的功能，在這章中將給您的是敍述的部份概念。

在我們開始之前，還要聲明一點，那就是雖然本章將介紹設計程式的概念，但却不能深入地涵蓋程式設計的內涵，如果您想要多瞭解程式設計的內涵，您可以參考各種BASIC語言的書籍。

說 明

我們討論BASIC敍述是由BASIC語言中電腦所不理會的敍述REM開始，這應該是適當的；當一個BASIC的敍述的起始三個字是REM，那麼電腦將不理會這個敍述所說的是什麼，而它的功能就是做一個說明（REMARKS），使您的程式易於閱讀。

如果您的程式只包含了五個或十個敍述，您也許很容易就可以記住這個程式的功能，但是如果您寫了一個有100個或200個敍述的程式，那麼您很可能在下一次使用它時就把這個程式許多重要的地方給忘了，尤其在您寫了許多的程式之後，您幾乎毫無疑問地無法記憶起每個程式的細節，而解決這個問題的方法，那就只有在您的程式中加上適當的說明，記錄這個程式的流程。

好的程式設計師在他們的程式中往往都會使用許多的說明；在本章中所有的程式範例我們都將使用說明，記錄程式的流程，希望可以使您養成將說明加入程式中的好習慣。

說明的敍述行與一般的敍述同樣地具有行號，而它們與其他敍述的行號並沒有任何的區別。

設定敍述

設定敍述（**ASSIGNMENT STATEMENTS**）使您可以將值放入變數內，在**BASIC**程式中您將時常碰到設定敍述。

以下就是一個設定敍述的例子：

```
 90 REM INITIALIZE VARIABLE X
100 LET X=3
```

在敍述100中，變數X中放入了3的值，這個敍述可以被改寫為：

```
100 X=3
```

LET 這個字是可以省略的，我們通常都不使用它。

下面是一個字串變數設定敍述的例子：

```
215 A$="ALSO RAN"
```

A$ 這個字串變數被放入了**ALSO RAN**這兩個字。

下面是三個我們在前面描述矩陣時所碰到的例子，這三個敍述將值放入矩陣變數**R$**( )內：

```
200 REM  R$() IS THE MOTEL GUEST LIST
210 R$(1)="JONES"
220 R$(2)="SMITH"
230 R$(3)="DOE"
```

記得一點，我們可以在同一行中放入不只一個的敍述；因此以上的三個敍述可以放在同一行中，就如同下面的情形：

```
200 REM   R$() IS THE MOTEL GUEST LIST
210 R$(1)="JONES":R$(2)="SMITH":R$(3)="DOE"
```

回憶一下，當同一行中出現不只一個的敍述時，它們之間必須使用冒號（：）做爲分界。

設定敍述可以包含本章中所介紹的任何算術及邏輯運算，以下就是一個例子：

```
 90 REM  THIS A DUMB WAY OF ASSIGNING A VALUE
100 V=33+7/9
```

這個敍述將值 33.7777778 放入實數變數 v 內，這與下面的三個敍述的功效相等：

```
 90 REM  X AND Y NEED TO BE INITIALIZED SEPARATELY FOR
 LATER USE
100 X=7
110 Y=9
120 V=33+X/Y
```

而這三個敍述也可以放在同一個程式行內，如下所示：

```
100 X=7:Y=9:V=33+X/Y
```

下面的兩個布爾運算的設定敍述是我們在本章前面所提過的：

```
 90 REM  THESE EXAMPLES WERE DESCRIBED EARLIER IN THE
 CHAPTER
100 A= NOT ((3+4)>=6)
110 B=("AA"="AB") OR ((8*2)=(4 ^ 2))
```

以下的例子顯示出在 APPLESOFT 中如何使用字串連接的方法將不同的字串實數連接在一起：

```
 90 REM   R$(6) IS ASSIGNED THE VALUE MR. ALTON
100 MR$ = "MR. "
110 MS$ = "MS. "
120 N$ = "ALTON"
200 R$(6) = MR$ + N$
```

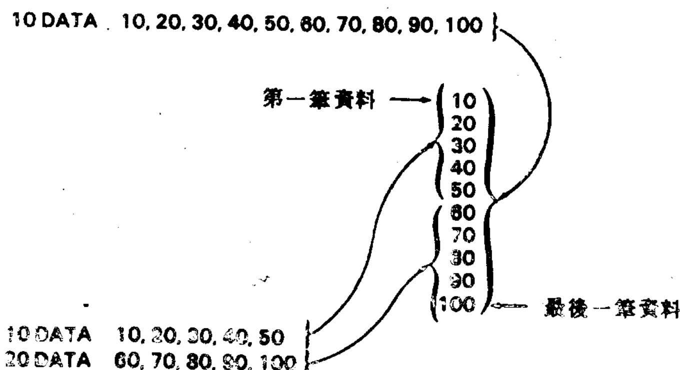
DATA及READ敍述

在 APPLESOFT 的程式中，當您有許多的變數須要存放入資料時，您可以不必使用前面所講的設定敍述的方法，而使用 DATA 及 READ 敍述代替它。考慮一下以下的例子：

```
 5   REM   INITIALIZE ALL PROGRAM VARIABLES
10   DATA   10, 20,-4, 300
20   READ A,B,C,D
```

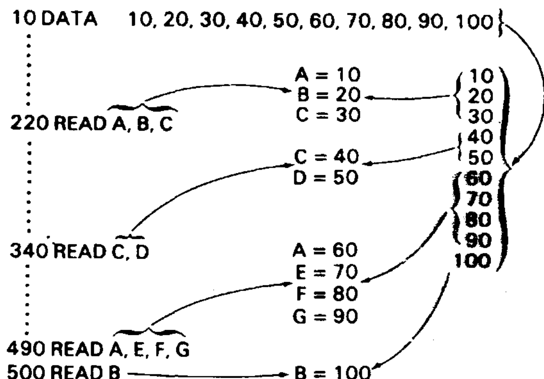
在行號 10 的這個敍述中指出了四個數值資料，而這四個數值資料在行號 20 的敍述中放入了四個數值變數中，在行號 10 及 20 的敍述被執行以後，我們得到 A= 10 ，B= 20 ，C=－4 ，D= 300 的結果。

如果在您的程式中有一個或更多的 DATA 敍述，那麼您可以將它們視爲連接成一行的值；例如，一個 DATA 敍述擁有 10 個值將建立一個擁有 10 個資料的行，而若有兩個 DATA 敍述，每一個都有上述 10 個值的其中前（或後）五個，那麼這兩個敍述相連接後所造成的效果是與只有一個敍述相同的，也同樣建立了一個有 10 個資料的行。下面就是一個說明：



第一個 READ 敍述從行的第一個值取起，依次將它們放入READ

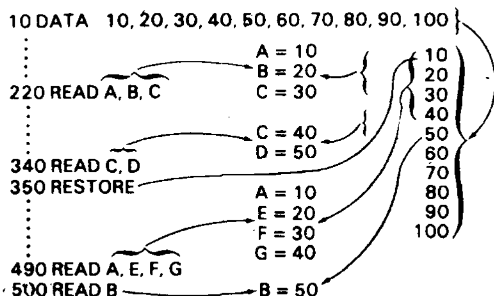
敍述中的變數名稱裏，而第二個 **READ** 敍述則從第一個 **READ** 敍述結束的地方開始，依次從行中取值。這種情形我們可以作如下的說明：



DATA 行可以擁有數值及字串值；當您使用 **READ** 敍述將 **DATA** 行中的值放入變數中時，變數及值必須是同一種的型式（字串或數值）。

RESTORE敍述

您可以藉著**RESTORE** 敍述將指標移到 **DATA** 行的起始處。下面就是**RESTORE** 使用的說明：



變數值的清除

整數BASIC及APPLESOFT都允許您將數值變數及矩陣的值及字串變數、字串矩陣的元素一次清爲∅或是空白。

在整數BASIC中，您可以在立即式的情況下使用CLR命令達成這個目的。下面就是一個例子：

```
>X=37

>PRINT X
37

>CLR

>PRINT X
0
```

而在APPLESOFT中，您就必須使用CLEAR敍述來完成這個工作，它同時將DATA行的指標移到開始處——就如同RESTORE所做的一般。以下是一個例子：

```
]10 REM INITIALIZE VARIABLES

]20 X=37

]30 A$="PIG IRON

]40 PRINT A$

]50 CLEAR

]60 PRINT X

]RUN
PIG IRON
0
```

宣告矩陣的容量及字串長度

如果您想在您的程式中使用矩陣或字串變數，您必須使用DIM敍

述在程式開始的地方宣告它們的最大容量或行列數；只要這個 DIM 敍述不超過一個程式行的容量，它可以同時宣告任何數目的矩陣及字串變數。

在整數 BASIC 中，當您做宣告的工作時，您將矩陣或字串變數的名稱在 DIM 敍述中寫出，接著用括弧括起它們的長度或最大容量，但是只有一階的數值矩陣是被允許的，至於字串矩陣或多階的行列矩陣並不被承認。以下的例子顯示出含有 5 個及 25 個字的字串以及含有 13 個元素的數值矩陣（ 0 到 12 ）：

```
10 DIM S1$(5),S2$(25),NB(12)
```

在 DIM 敍述中跟隨在字串變數後面的數字指出這個字串在該程式中所能包含字的數目的極限，而在數字矩陣名稱後面的數字就等於您可以在這個矩陣中使用的最大腳註的數目。

在 APPLESOFT 中，當它第一次碰到矩陣時，它將檢查您是否曾經宣告過這個矩陣，如果您並沒有宣告它，那麼 APPLESOFT 將提供每一行列 0 到 10 的腳註，所以如果您所要使用的矩陣的行列都不超過 11 個腳註的話，您不須要先宣告它。以下的例子宣告了一階的矩陣R$ ，同時也宣告了具有 21 個元素的整數矩陣：

```
]15 DIM R$(10),R%(20)
```

我們在前面所描述的旅舍房客的例子，可以用以下的方式來完成這個二階矩陣的宣告：

```
]15 DIM H$(8,10)
```

跟隨在矩陣名稱後面的數目也就是在該腳註位置所能容許的最大腳註數目，但您必須注意的是由 ∅ 開始的，因此R$（10）使R$（ ） 這個變數擁有 11 個值而非 10 個，因為腳註可以分別是 ∅、1、2、3、4、5、6、7、8、9 及 10 ，而H$（8，10）同樣地就可以擁有 99 個元素，因為它的第一階腳註可以是 ∅、1、2、3……及 8 ，而第二

階的脚註可以由0到 10 。

重訂矩陣的行列數

除非您重新執行這個程式，否則您若想重訂其中矩陣的行列數是不可能的；每一個脚註的值都必須在0與該矩陣宣告的脚註數目之間，任何超出的脚註值都將被認爲是不合法的。

轉向敍述

在BASIC程式中，敍述的執行先後順序是依照行號的大小而排列的，這種情形我們曾在本章的前半段講過，轉向敍述（BRANCH STATEMENTS ）改變了這個執行的順序。

GOTO 敍述

GOTO 是最簡單的轉向敍述，它允許您指出下一個要執的敍述。考慮以下的例子：

```
 20   A = 4
 30   GOTO 100
 40
 50
 60
 70
 80
 90
100
110
等等
```

行號 20 的程式行是一個設定敍述，它將一個數字放入變數A中，下一個敍述就是 GOTO ，它指出這個程式的執行必須轉移到行號100的地方去，因此在這個程式的第一部份的執行順序是：行號 20 ，然後是行

號 30 ，接著就是行號 100 。

當然，其他的一些敍述必須將程式的執行轉移到行號 40 的地方去，否則依照以上的邏輯，行號 40 將永遠執行不到了。

您可以將程式的執行轉移到任何行號去，即使是一個說明（ REM ）的程式行也沒關係，電腦將不理會它，所以像這樣的轉移就如同轉移到這程式行的下一個程式行一般。例如，考慮以下的轉移：

```
20  A = 4
30  GOTO 70
40
50
60
70  REM THIS LINE CONTAINS ONLY A REMARK
80
90
etc.
```

程式的執行從行號 30 轉移到行號 70 ，但是在行號 70 上只有說明，因此電腦就自動地執行行號 80 的敍述；因此，雖然您可以將程式的執行轉移到說明的地方，但您最好還是將程式的執行轉移到它的下一行，以免浪費執行的時間。下面是一個解說：

```
20  A = 4
30  GOTO 80
40
50
60
70  REM THIS LINE CONTAINS ONLY A REMARK
80
90
etc.
```

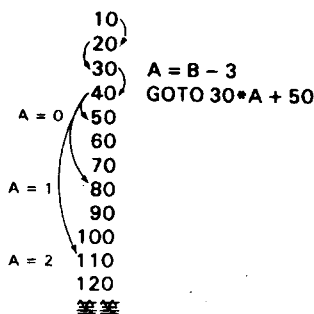
想要將程式的執行轉移到一個不存在的行號將會導致錯誤的發生。

計算式的 GOTO 敍述

還有另一種的 GOTO 敍述使得程式的執行依照數值運算式的結果

而轉移到幾個程式行中的一個去執行。

考慮下面這個整數BASIC敍述：



行號 40 就是一個計算式的 **GOTO**（**COMPUTED GOTO**）敍述，當這敍述被執行時，程式將轉移到該計算式所得到的結果的那一個程式行上；在這個例子中，如果變數 **A** = 0 ，那麼程式的執行將轉移到行號 50 的敍述上去，或是 80 的敍述上（當**A** = 1 ），而當**A** = 2 時，程式的執行將轉移到行號 10 的敍述。如果計算式所得出的結果並不包含在該程式的程式行號中，那麼您就將得到 *** **BAD BRANCH ERR** 的錯誤訊息。需要注意的是，變數**A**在敍述 30 中被給了一個值，而**A**的值依賴變數**B**的值而定，上面的例子並沒有說明變數**B**的值是如何得到的，但**B**的值只要是 3 、 4 或 5 ，那麼行號 40 的敍述就會使轉移發生。

想要測試整數**BASIC**中的**GOTO**敍述，您可以輸入以下的程式:

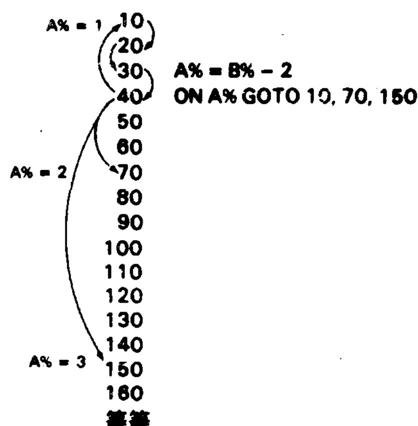
```
>9  REM  INITIALIZE VARIABLE B
>10  B = 4
>20  PRINT B
>30  A = B - 3
>40  GOTO 30 * A + 50
>49  REM  B = 3
>50  END
>79  REM  B = 4
>80  PRINT B
>90  B ='5
>100  GOTO 20
```

```
>109   REM  B = 5
>110   PRINT B
>120   B = 3
>130   GOTO 20
```

現在打入 **RUN** 指令執行這個程式。

您能瞭解數字顯示情況的原因嗎？試著改寫這個程式，使得每個數字依照下面的順序只顯示一次：345345……

APPLESOFT 的計算式 **GOTO** 敍述與整數 **BASIC** 有一些不同的地方，如同以下所顯示的。



在行號 40 的敍述就是 **APPLESOFT** 中的計算式 **GOTO** 的型式，當這敍述被執行後，程式的執行將依照A％的值而分別轉移到行號 10 的敍述（A％＝1）、行號 70 的敍述（A％＝ 2）或行號 110的敍述（A％＝3）上，若A％的值並非1、2或3，那麼程式的執行仍然到行號 50 的敍述上去。

APPLESOFT 中計算式 **GOTO** 敍述中的運算式的值決定轉移的行號數目，當它的值是 1 時，那麼第一個行號將被使用，若它的值是 2 的話，則使用第二個行號作爲轉移的對象，依此類推。若運算式的值是 0 或超出了這個敍述所具有的行號的數目，那麼電腦將繼續執行該計

算式 GOTO 的下面一個敍述。

以下的例子顯示出在 APPLESOFT 中計算式 GOTO 的執行情形。

```
10 B% = 4
20  PRINT B%
30 A% = B% - 2
40  ON A% GOTO 10,70,150
70  PRINT B%
80 B% = 5
90  GOTO 30
150  PRINT B%
160 B% = 3
170  GOTO 20
```

迴　圈

GOTO 及計算式的 GOTO 敍述使您能夠隨心所欲地改變程式執行的順序；但是假設您想要重複執行同一個命令或一組的命令許多次；例如，假設一個矩陣變數A（ I ）有 100 個元素，而每一個元素須要給它一個從 0 到 99 的值，那麼寫100 個設定敍述去做這件事將是非常累人的。那麼使用一個迴圈（ LOOP ）並執行它100 次不是簡單得多嗎?

FOR及 NEXT敍述

您可以仿照以下的例子使用 FOR 及 NEXT 敍述創造一個迴圈：

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 NEXT I
```

在 FOR 及 NEXT 之間的敍述將重複地被執行，在以上的情況下，一個設定敍述出現在 FOR 及 NEXT 之間，因此這個敍述就被重複地執行了；這種型式的程式結構就被稱爲 FOR-NEXT 迴圈。

所以您可以由以下的例子看出 FOR-NEXT 迴圈的執行情形，這個程式將矩陣 A( )內的值顯示出來。

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

當您打入 **RUN** 指令後，這個程式將把 0 到 99 共 100 個數字顯示出來。

在 **FOR** 及 **NEXT** 中間的敍述所執行的次數由緊跟在 **FOR** 後的脚註變數（ **INDEX VARIABLE** ）所決定；如上面例子它的脚註變數就是 I ，而 I 的值是由 0 到 99 每次增加 1 。

變數 I 同樣的出現在行號 30 的設定敍述中，因此當這個設定敍述第一次被執行時，I 的值就是 0 ，而這個設定敍述就相當於：

```
30 A(0)=0
```

I 依照增加幅度（ **STEP SIZE** ）而增加它的值，在行號 20 中我們看到是 1 ，因此當行號 30 的設定敍述第二次被執行時，I 的值就是 1 ，而這個設定敍述就變成：

```
30 A(1)=1
```

I 值繼續依照增加幅度而增加，直到極大值 99 時爲止（或超過 99 ）。

增加幅度並不須要一定是 1 ，它可以是任何整數值。改變行號 20 中的增加幅度爲 5 ，再重新執行這個程式，現在這個設定敍述將只被執行 20 次，因爲將 I 的值每次增加 5 ， 19 次之後就得到 95 了，第 20 次的增加將使 I 的值等於100，那就已經超過了它的極大值 99 。若想使這設定敍述重複被執行 100 次而增加幅度仍然是 5 ，您只須將 I 的極大值增加到 500（記得同時也需要將 **DIM** 敍述更改）。

增加幅度可以不必是正數，但是如果增加幅度是負值，那麼 I 的起始值必須較它的最後值爲大。例如，如果增加幅度是－1 而我們想要將 0 到 99 的數值分別放入 A(I) 的 100 個元素內，那麼我們必須重寫行號 20 的敍述：

```
10 DIM A(99)
20 FOR I=99 TO 0 STEP -1
30 A(I)=I
35 PRINT A(I)
40 NEXT I
50 END
```

請執行這個程式以便測試負值的增加幅度。

如果增加幅度是1（這是經常碰到的事），您不必定義增加幅度，因爲BASIC將自動地假設增加幅度爲1。

您也可以使用運算式來定義脚註的起始、末尾值及增加幅度，但您應該儘量避免這樣做，因爲如此只是增加了程式不必要的複雜性罷了！如果您必須計算其中之一的值，那麼最好就在迴圈之前先將這個計算完成。

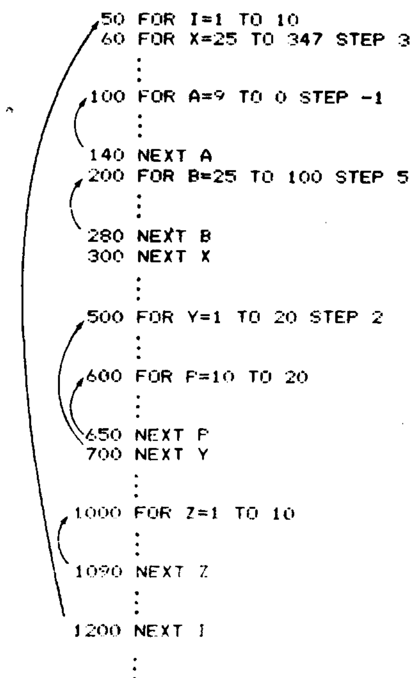
在 **APPLESOFT** 中您可以在起始值、末尾值及增加幅度中使用實數值；在 **NEXT** 敍述中，您不必指出脚註變數；但如果您做了，它將使您的程式容易閱讀。

多重結合的迴圈

FOR-NEXT 這樣的結構被稱爲程式迴圈（**PROGRAM LOOP**），因爲敍述的執行由 **FOR** 開始，到 **NEXT** 結束，然後又回到 **FOR** 的地方，像這樣的迴圈結構可以說是非常的普遍的，幾乎您所寫的**BASIC**程式都將包含一個或更多像這樣的迴圈；由於迴圈是這樣地普遍，因此它們時常地被包含在其他性質相同的迴圈內；在 **FOR** 與 **NEXT** 之間經常可能有 10 個、100 個或任何數目的敍述，而在這些敍述內，可能也包含了其他的迴圈，下面的例子顯示了單層結構的迴圈:

```
10 DIM A(99)
20 FOR I=0 TO 99 STEP 1
30 A(I)=I
40 REM  DISPLAY ALL VALUES OF A(I) ASSIGNED THUS FAR
50 FOR J=0 TO I
60 PRINT A(J)
70 NEXT J
80 NEXT I
90 END
```

即使在一個很短的程式內，複雜的迴圈結構也可能經常出現，下面就是一個例子，表示多重的 **FOR** 及 **NEXT** 敍述情形：



```
50 FOR I=1 TO 10
60 FOR X=25 TO 347 STEP 3
   :
100 FOR A=9 TO 0 STEP -1
   :
140 NEXT A
200 FOR B=25 TO 100 STEP 5
   :
280 NEXT B
300 NEXT X
   :
500 FOR Y=1 TO 20 STEP 2
   :
600 FOR P=10 TO 20
   :
650 NEXT P
700 NEXT Y
   :
1000 FOR Z=1 TO 10
   :
1090 NEXT Z
   :
1200 NEXT I
   :
```

最外層的迴圈使用脚註變數**I**，而這個迴圈包含了三個子迴圈，它們分別使用**X**、**Y**、**Z**爲它們的脚註變數；在迴圈**X**以內又包含了兩個子迴圈，它們分別使用**A**、**B**做爲脚註變數，而在**Y**迴圈內也包含了一個子迴圈，它則使用**P**做爲它的脚註變數。至於迴圈**Z**則不包含任何的子迴圈。

迴圈的結構是非常容易瞭解及使用的；這裏只有一個經常發生的錯誤您必須避免，那就是決不要在迴圈的子迴圈結束之前，結束這個迴圈；例如，以下的迴圈結構就是不合法的：

```
 50 FOR I=1 TO 10
 60 FOR X=25 TO 347 STEP 3
     :
100 NEXT I
     :
200 NEXT X
```

每一個程式中都必須包含同樣數目的 **FOR** 及 **NEXT** 敍述，因爲每個迴圈都必須由 **FOR** 開始，而在 **NEXT** 敍述結束。

例如，假設程式中擁有一個 **FOR** 敍述，而擁有兩個 **NEXT** 敍述，那麼這第一個 **NEXT** 敍述與 **FOR** 敍述配對做了正確的執行動作，但是對第二個 **NEXT** 而言，由於沒有 **FOR** 敍述與它配對，於是就產生了錯誤。

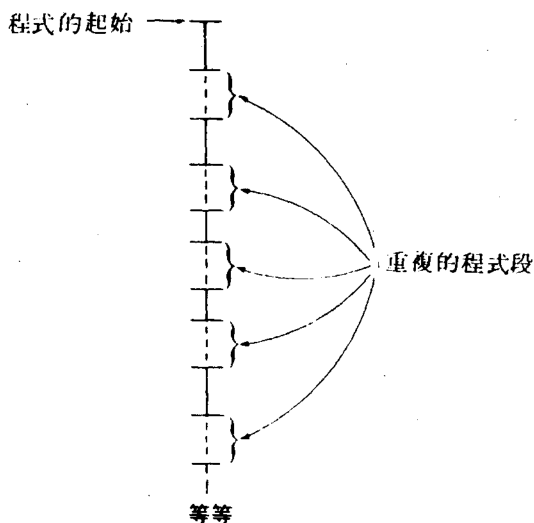
如果您在 **APPLESOFT** 的程式中，使用 **NEXT** 敍述而沒有包含腳註變數在內，程式依然會正確地執行，因爲當 **NEXT** 敍述被碰到時只有一種正確的結束迴圈的方法。如果您對這一點並不相信，您可以試著寫一些多重結構的迴圈試試。

副程式的敍述

當您開始寫包含較多程式行的程式時，您將很快地發現某些短短的一段程式經常被重複地使用到。例如，假設您的程式中包含有一個矩陣變數（例如A()），而它的值經常在程式的許多地方重新被定義，那麼您可能會使用 **FOR** 及 **NEXT** 敍述重複地定義這個矩陣變數，由於這麼做只包含了三個指令，您也許並不嫌麻煩。

但是假設在定義這個矩陣的迴圈內還包含了 10 個或 11 個的敍述以便使這個矩陣具有某種特性，那麼當您重複地將這個迴圈加入在程式的不同地方時，那不但浪費您的時間，而最重要的是它們佔用了許多的記憶體位置，這在下面的圖形中做了一個說明。

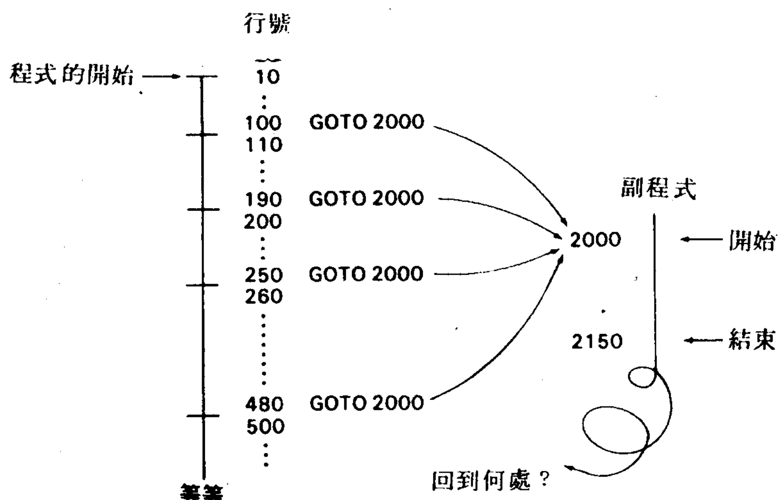
那麼如果我們將這些重複的敍述與程式分開，然後在需要使用它們時再將程式的執行轉移過去，是不是一個好的解決方法？



而這正是我們所要做的工作，至於那一組敘述我們就稱它們做副程式（SUBROUTINE）。

現在又有一個問題發生了，那就是把程式的執行轉移到副程式是一件簡單的工作——因爲副程式有一個開始的行號，但是在副程式結束之後，您如何將程式的執行轉移回到主程式去呢？

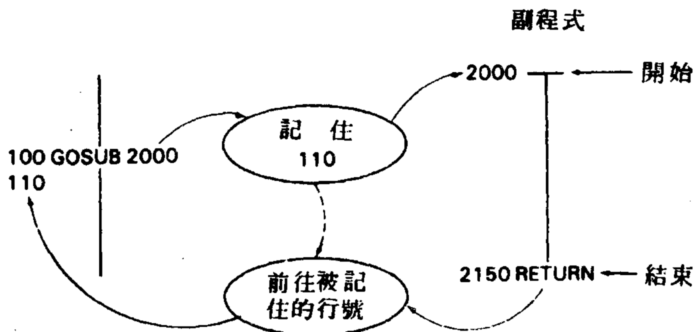
您可以藉著使用 GOTO 敘述將程式的執行轉移到副程式去，下面就是一個說明的例子：



但是在副程式結束後，程式的執行轉移到何處？如果有兩個或更多的 **GOTO** 敍述將程式的執行轉移到副程式，那麼就有兩個或更多的程式行是您在副程式結束後希望將程式執行轉移到那裏去的，而這解決的方法就是使用一個特殊的副程式敍述 **GOSUB** ，而不是使用 **GOTO** 來做程式執行的轉移工作。

GO SUB敍述

這個敍述如同 **GOTO** 一般地達成執行轉移的功能，但它却把執行完副程式後要轉回的地方記住；在電腦術語中，我們說 **GOSUB** 就是呼叫（ **CALL** ）一個副程式。下面就是它的說明：



在副程式結束的地方，我們加上**RETURN**敍述，這個敍述使得程式的執行轉移回到 **GO SUB** 敍述下面的一個敍述上，如果 **GO SUB** 是這個程式行的最後一個敍述，那麼程式執行將轉移到下一個程式行的第一個敍述去。

我們前面提過可以使用三個敍述的迴圈來定義矩陣A( )，如果我們將它變成一個副程式，那麼就會像下面的程式一般：

```
  10 REM    MAIN PROGRAM
  20 REM    YOU CAN DIMENSION A SUBROUTINE'S
  30 REM    VARIABLE IN THE MAIN PROGRAM.
  40 REM     IT IS A GOOD IDEA TO DIMENSION ALL
  50 REM    VARIABLES AT THE START OF THE MAIN PROGRAM
  60 DIM A(99)
  70 GOSUB 2000
  80 REM    DISPLALY SOMETHING TO PROVE THE RETURN OCCURRED
  90 PRINT "RETURNED"
 100 END
2000 REM    SUBROUTINE
2010 FOR I=0 TO 99
2020 A(I)=I
2030 PRINT A(I)
2040 NEXT I
2050 RETURN
```

POP敍述

在某些情況下您也許不希望在副程式執行結束後，程式的執行轉移到 GOSUB 之後的第一個敍述去，您也許想要使用 GOTO 敍述做這個轉移，但是由於BASIC語言會記住副程式結束後應該返回的地方，因此這也不能成功地達成您的需求。在這種情形下，您就可以使用POP敍述，否則您就會由於RETURN敍述沒有被執行而得到錯誤訊息。POP 敍述的功能就是它會使BASIC語言忘記掉最接近的那個應該返回的位置，因此您就可以使用 GOTO 敍述將程式的執行轉移到您希望的地方去了。

您在將RETURN敍述忽略掉時最好小心從事，因爲過份地使用 POP 敍述而使得 GOTO 敍述得以將程式的執行轉移到副程式之外的地方，常常會使您的程式糾纏不清而易於混淆。

多重結合的副程式

副程式也是可以多重結合的，那也就是說，一個副程式可以呼叫另一個副程式，而第二個副程式又可以呼叫第三個副程式，如此繼續下去。在您想要使用多重副程式時，您不必多做任何特別的工作，您只需要使用 GOSUB 去呼叫這個副程式，當然這個副程式必須使用RETURN 敍述結束它自己的動作，BASIC語言將記住當每一個副程式結束後應該轉移的行號。

下面的例子就是多重結合副程式的一個說明：

```
10 REM    MAIN PROGRAM
20 REM    YOU CAN DIMENSION A SUBROUTINE'S
30 REM    VARIABLE IN THE MAIN PROGRAM.
40 REM     IT IS A GOOD IDEA TO DIMENSION ALL
50 REM    VARIABLES AT THE START OF THE MAIN PROGRAM
60 DIM A(99)
```

```
  70 GOSUB 2000
  80 REM   DISPLAY SOMETHING TO PROVE THE RETURN OCCURRED
  90 PRINT "RETURNED"
 100 END
2000 REM  FIRST LEVEL SUBROUTINE
2010 FOR I=0 TO 99
2020 A(I)=I
2030 GOSUB 3000
2040 NEXT I
2050 RETURN
3000 REM  NESTED SUBROUTINE
3010 PRINT A(I)
3020 RETURN
```

這個程式只是將從行號 2000 開始的副程式內的**PRINT A(I)**敍述移到位於行號 3000 的多重結合副程式內，其他的都沒有改變。

對於一個副程式可以呼叫另一個副程式這種情形是被認爲可以接受而且極爲需要的，但是一個副程式絕對不可以呼叫它本身，至於一個副程式呼叫另一個副程式，而被呼叫的這個副程式又呼叫原先呼叫它的副程式，也是不被允許的，這種情形被稱爲遞歸（**RECURSION**），在**APPLE Ⅱ** 的**BASIC** 語言中是不可以使用的。

計算式的GO SUB敍述

GOTO 與 **GOSUB** 的邏輯非常類似，它們之間唯一的差別只在於 **GOSUB** 記得下一個行號，因此您對於計算式的 **GOSUB** 敍述應該不會有什麼詫異才對；計算式的 **GOSUB** 敍述允許您依照數值運算式的值而將程式的執行移轉到許多個副程式中的一個，而這計算式的 **GOSUB** 敍述會將所要返回程式行的行號記住，至於那一個副程式被呼叫並沒有任何的影響，被呼叫的副程式中的**RETURN**敍述將會使程式的執行轉移到所被記住的程式行去。

您可以使用計算式的 **GOSUB** 敍述將副程式多重結合起來，就如同您使用標準的 **GO SUB** 敍述將副程式多重結合一般。

考慮下面的整數**BASIC** 敍述：

```
>100 GOSUB A*500+2000
>110 REM
```

在行號100的程式行中就是一個計算式的 GOSUB 敍述，當這個敍述被執行時，程式的執行將依照運算式的結果而轉移到適當的行號上去。在這個例子中，如果A＝ø，那麼將轉移到行號2000的敍述上去，若A＝1則程式執行就被轉移到行號2500的敍述上去了。如果運算式計算得到的行號並不存在程式中，那麼您將得到 ***BAD BRANCH ERR 的錯誤訊息。

在 APPLESOFT 中，計算式的 GOSUB 敍述就如同計算式的 GOTO 敍述一般的運作。以下是一個例子：

```
]90
]100 ON A GOSUB 1000,500,5000,2300
]110 REM
```

當行號100的敍述被執行時，如果A＝1則從行號1000開始的副程式就被呼叫，當A＝2時，則從行號500開始的副程式就是程式執行轉移的地方，A＝3時，起始行號5000的副程式就被呼叫，至於A＝4時，那麼由行號2300開始的副程式就是被呼叫的對象了。如果A的值並非1、2、3、4那麼電腦將繼續依次執行行號110的程式行（沒有任何副程式被呼叫）。

有條件地執行

計算式的 GOTO 及計算式的 GOSUB 都是條件式的敍述（CONDITIONAL STATEMENTS），這也就是說，程式執行的流程依賴一個或更多個能夠在程式執行中被改變的值而決定；因此確實的程式流程就是依賴著這些變數的情況而定。

IF-THEN敍述

另外一個條件式的敍述是 IF - THEN 敍述，它具有以下的型式：

IF 運算式 THEN 敍述

如果運算式是眞，那麼 THEN 後面的敍述就被執行；相關及布爾運算式是最常被使用在 IF - THEN 敍述中的，當然算術運算式也可以同樣地被使用，這使得BASIC程式具有做決定（DECISION-MAKING ）的能力。以下是一些 IF - THEN 敍述的例子：

```
10 IF A=B+5 THEN PRINT MSG$
40 IF CC$="M" THEN IN=0
50 IF Q<14 AND M<M1 THEN GOTO 66
```

在行號 10 的PRINT敍述可能會被執行，如果變數A的值等於B的值加 5 的話；否則這個PRINT敍述將執行不到。

如果 CC$ 字串裏面的字是M的話，那麼在行號 40 的敍述將把0放入數值變數 IN 內。

如果變數Q的值小於 14 而且變數M的值較M 1 的值小的話，行號 50 的敍述將使程式的執行轉移到行號 66 的程式行去。否則，程式的執行將繼續往下一個程式行移動。

如果您對 IF 後面運算式的運算並不瞭解的話，那麼請您參考本章前面討論運算式的地方。

IN - THEN 敍述可以在同一個程式行中跟著其他的敍述，然而整數BASIC與 APPLESOFT 對於處理這種情形却稍有不同。

在整數BASIC中，只有緊跟在 THEN 後面的敍述是有條件的被執行的，至於在同一個程式行中的其他敍述則不論在 IF - THEN 敍述中的運算式是眞或假，它們都將被執行。這可以由下面的例子做一個解說：

```
10 IF V>100 THEN PRINT "DEWEY WINS": GOSUB 2000
20 T=T+V: PRINT T
```

在上面的這個例子中，這個程式只要V的值大於100，那麼將會印出DEWEY WINS 的訊息，但是不論V的值是多少，這個程式都將呼叫在行號 2000的副程式。

至於 APPLESOFT 對於跟隨在 IF-THEN敍述後面而且在同一個程式行的敍述而言，只有在 IF-THEN敍述的運算式是眞，它們才會被執行；如果運算式的結果是假，那麼程式的執行將轉移到下一個程式行的第一個敍述去；就上面的這個例子在 APPLESOFT 中而言，只有在V的值大於 100時，程式才會把DEWEY WINS 的訊息印出，並且呼叫在行號 2000 的副程式，如果V値是小於或等於 100，那麼這個程式將不會印出任何訊息，而且也不會去呼叫副程式，程式的執行將轉移到在行號 20 的第一個敍述去。

在 APPLESOFT 中 IF-THEN敍述有一個特別的格式是可以供您使用的，當有條件被執行的敍述是 GOTO 敍述時，您可以省略掉 THEN 字；以下的兩個敍述是可以互換的：

```
]10 IF MM$=DD$ THEN GOTO 100
```

它與以下的程式行相同

```
]10 IF MM$=DD$ GOTO 100
```

輸入與輸出敍述

有許多的BASIC敍述控制著資料輸入及輸出電腦的工作，我們總稱它們爲輸入/輸出敍述（INPUT/OUTPUT STATEMENTS）。最簡單的輸入/輸出敍述控制著資料由鍵盤輸入及由顯示螢光幕輸

出，我們將在接下去的部份討論這些簡單的輸入／輸出敍述。當然我們還有比較複雜的輸入／輸出敍述控制著在電腦與週邊裝置間的資料轉移工作；例如，磁帶機、磁碟機及列表機，這些較複雜的輸入／輸出敍述我們將在第四章及第五章加以描述，在第六章中我們將討論能使圖形顯示在螢光幕上的輸出敍述。

我們已經在前面使用PRINT敍述，它的功能就是把資料輸出到顯示在螢光幕上，所以在我們討論輸入敍述之前，先看看這個敍述。

PRINT敍述

爲什麼我們使用PRINT而不使用DISPLAY或DISPLAY的縮寫字來代表DISPLAY的意義呢？這答案是因爲在六〇年代，當BASIC程式語言才被創造時，顯示螢光幕的價格非常高昂，因此不適於做爲較便宜的電腦週邊裝置；而一個標準的電腦就具有主機及列表機，所有的資料都是被列印到紙上而不是顯示在螢光幕上；因此從那時開始就一直使用PRINT敍述來描述在螢光幕上顯示資料的功能。

PRINT敍述可以顯示字或是數字；例如，以下的敍述將顯示出TEXT這個字：

```
10 PRINT "TEXT"
```

想要顯示出數字，您將數字或一個變數名稱放在PRINT的後面，就如同下面的情形：

```
>A=10
>PRINT 5,A
5          10
```

以上的敍述將數字5及10顯示在同一行中。

您可以在PRINT敍述之後使用混合型式（同時具有文字及數字）

的資料，使它們顯示在螢光幕上，當然在使用許多不同項目的時候；您必須使用逗號將它們逐一地分開。以下的PRINT敍述顯示出ONE、TWO、THREE、FOUR及FIVE這些字，並跟隨著它們各自相對應的數字。

```
10 PRINT "ONE",1,"TWO",2,"THREE",3,"FOUR*,4,"FIVE",5
20 END
```

當您使用逗號來分離變數時，那麼APPLE Ⅱ在顯示這些變數時將自動地預先設定好一定數目的空白來分離這些變數的值；您可以藉著執行上述的例子印證這種情形。如果您想要將變數中間間隔的空白省略掉，那麼您就使用分號來分離在PRINT敍述中的變數，就如同下面的情形：

```
10 PRINT "ONE";1;"TWO";2;"THREE";3;"FOUR";4;"FIVE";5
20 END
```

您可以再次的輸入以上的敍述藉以印證分號的功能。

PRINT敍述當它被執行完畢後，會自動地將游標移到下一行的最左邊位置去；在電腦術語中，這種情形就被稱爲游標回頭（CARRIAGE RETURN）。

您可以在PRINT敍述的最後一個值的後面加上一個逗號或分號將游標回頭的功能消除掉。跟在最後一個值後面的逗號會把游標移到下一個值所要顯示的位置去。我們藉著下面擁有三個敍述的程式來說明，在您打入這個程式後再輸入RUN的命令：

```
10 PRINT "ONE",1
20 PRINT "TWO",2
30 END
```

現在把逗號加在行號 10 的程式行後面並輸入 RUN 命令，您將看到行號 20 敍述的輸出緊接在行號 10 敍述輸出的後面。

現在把逗號換成分號並重新執行這個程式，您將發現這兩個程式行的輸出仍然連接在一起，只是數值1與TWO這個字之間的空白已不再存在了。藉著分號的使用可以使變數之間的空白消除掉，這是一個供您選擇使用的功能。

我們已經把直接加在PRINT敍述中的數字從螢光幕上顯示出來了，現在您也可以將變數內所包含的值同樣地顯示在螢光幕上。以下的程式與第一個PRINT敍述做同樣的工作，只是使用矩陣A( )存放數字；試著輸入這個程式並執行它：

```
 5 DIM A(5)
10 FOR I=1 TO 5
20 A(I)=I
30 NEXT I
40 PRINT "ONE";A(1);"TWO";A(2);"THREE";A(3);"FOUR";A(4);
 "FIVE";A(5)
50 END
```

在 APPLESOFT 中，您可以將所要顯示的字放在字串變數中，並把PRINT敍述放在 FOR-NEXT 的迴圈內，那麼上面的程式可以改爲以下的型式：

```
10 DATA  "ONE","TWO","THREE","FOUR","FIVE"
20 FOR I = 1 TO 5
40 READ N$
50 PRINT N$;I;
60 NEXT I
70 END
```

INPUT敍述

當一個INPUT敍述被執行的時候，電腦將等待著您由鍵盤輸入資料，在電腦得到您輸入的資料之前，它將不做任何其他的動作。

INPUT 敍述最簡單的型式就是在INPUT字後面接著一個變數的名稱，由鍵盤輸入的資料就被放在這個變數名稱裏面；輸入資料的型

式必須與變數名稱的型式相符合，例如，一個數值變數只能接受數值的輸入資料。您可以打入以下的程式並執行它，藉以瞭解數值輸入資料的情形（您也可以試著輸入一些字的資料，並看看它執行的結果如何）：

```
10 INPUT A
20 PRINT A
25 REM  END PROGRAM IF 0 ENTERED
30 IF A = 0 THEN  END
40 GOTO 10
```

電腦執行INPUT敍述時，在顯示幕上將會顯示出一個問號，等待著您的資料輸入；上面的程式將您所按的每一個鍵顯示出來；在電腦的術語中，我們稱這種情形爲顯示幕反應（ECHOES）鍵盤。由於在行號 20 處的PRINT敍述使得您輸入的數目顯示出來；上面程式的第一個顯示是由於行號 10 的敍述被執行的結果，而同時電腦等待著您輸入資料，至於第二個顯示則是由於行號 20 的PRINT敍述執行的結果。

一個INPUT敍述可以同時接受不只一個值；您可以在INPUT敍述後面，接連著列出您想要接受值的一連串變數名稱，並且以逗號分開它們，您就可以達成這個目的了。當像這樣的INPUT敍述被執行時，您必須對應著每一個變數給它們各別的值；需要注意的是值與變數的型式必須一致。

當您在回應INPUT敍述時，您千萬不要在很大的數目中使用逗號做爲標點；例如，在輸入 1000 時絕不要用 1,000。

以下的例子輸入兩個數值並將它們顯示在螢光幕上：

```
20 INPUT A,B
30 PRINT A,B
35 REM    END PROGRAM IF 0 ENTERED
40 IF A=0 OR B=0 THEN END
50 GOTO 20
```

執行這個程式並試著輸入兩個中間隔著逗號的數字，再按下RETURN鍵；現在試著只輸入一個數字並按下RETURN，如您所見到的，APPLE Ⅱ將提醒您輸入第二個值，這時您就再輸入一個值並按下

RETURN鍵。所以，當一個INPUT敍述需要不只一個數值作輸入時，您可以選擇一次在同一行中輸入所有的資料或是分別在幾行中輸入。

在整數BASIC中，INPUT敍述被使用在字串變數時，有少許的不同；第一點，它將不會顯示問號。試試看下面的例子：

```
10 DIM A$(19)
20 INPUT A$
30 PRINT A$
35 REM  END PROGRAM IF NULL ENTRY
40 IF A$="" THEN END
50 GOTO 20
```

當您執行上面這個程式時，試著輸入超過 19 個字的字串，您將得到 *** STR OVFL ERR 的訊息，而程式也將停止執行；您輸入的字串的長度絕不可以超過INPUT敍述中所能接受的最大的字串長度。

整數 BASIC 中，您必須使用一行做為輸入一個字串資料的位置；如果一個INPUT敍述內包含了一連串的變數，而這些變數中包含了字串變數，那麼在輸入字串變數資料時，必須單獨地使用一行的位置，這是因爲整數BASIC允許您將逗號包含在字串值之內的原因；您可以藉著執行以上的程式並且輸入DOE，JOHN的字串值印證這一點。以下的例子說明在整數BASIC中如果一個字串變數包含在INPUT敍述內將發生怎麼樣的結果。您可以做一個試驗：試著輸入四個值，並且都用逗號分開它們，看看結果如何？或是試著將它們分別用佔用一行的位置輸入。如果您在字串值內加入數值或逗號，結果又是如何？

```
10 DIM A$(10),B$(10)
20 INPUT A$,A,B$,B
30 PRINT A$,A,B$,B
35 REM  END PROGRAM IF NULL ENTRY
40 IF A$="" THEN END
50 GOTO 20
```

如我們前面討論的，在 APPLESOFT 中，一個實數變數可以擁有一個整數值；因此，您可以將整數值輸入給實數變數，而一個實數值

輸入給整數變數時將被轉變成整數，而轉變的原則您可以在本章中“混合型式的運算式”那一節中看到。

有提示的INPUT敘述

INPUT敘述本身是非常特別的，它的語法（SYNTAX）對於一般平常的操作人員而言似乎要求過多了一些；您試想一個對程式設計毫無所知的工作人員，當他由於輸入資料時不能夠將逗號的位置放對而得到了錯誤的訊息，那麼很可能發生的情況就是他沮喪地放棄了輸入資料的工作，而您可能因此需要花許多的時間去設計一些程式，這些程式的目的在於監視每一種操作員輸入資料時所可能發生的錯誤，而能夠使操作員瞭解他的錯誤所在。在第四章中我們將詳細討論這些資料輸入程式設計的技術。

INPUT敘述具有一個小小的功能使得您可以描述您所期待接受的值，而這種描述是藉著顯示在螢光幕上的一小段訊息來達成的，像這樣的訊息就被稱爲提示訊息（PROMPT MESSAGE），這個訊息在INPUT敘述中就是一串字用引號括起來，這個訊息將顯示在問號的前面；這在一連串的變數同時在一個INPUT敘述中時並不適用，它僅是提醒您何時應該輸入下一個資料而已。

在整數BASIC中，您必須將提示訊息緊接在INPUT字的後面，訊息的後面跟著一個逗號，接著就是一連串的變數；當同一個INPUT敘述中含有不只一個的變數時，提示訊息也只顯示一次，而且只在第一行輸入資料的地方。如果第一個變數是數值變數，那麼在提示訊息之後就會出現一個問號；若是第一個變數是字串變數，那麼就不會出現任何的符號。下面就是一個例子：

```
10 DIM A$(10)
20 INPUT "ENTER YOUR NAME AND AGE ",A$,A
30 PRINT A$;"IS";A
35 REM  IF ENTRY IS NULL, END PROGRAM
40 IF A$="" THEN END
50 GOTO 20
```

在 APPLESOFT 中，您在INPUT之後緊接著放入提示訊息，在提示訊息的後面接著分號，分號後面才跟著變數；由於使用了提示訊息的原因，INPUT敍述所產生的問號就被消除了；不管是有多少資料需要輸入提示訊息只出現一次。下面就是一個 APPLESOFT 的例子:

```
20   INPUT "ENTER YOUR NAME AND AGE:";A$,A
30   PRINT A$;" IS ";A
35   REM  USE CTRL-C TO END
50   GOTO 20
```

GET敍述

GET 敍述只能在 APPLESOFT 中使用，它可以由鍵盤接受一個字的輸入，但並不在螢光幕上顯示出來，當您打入這個字的時候您並不需要按 RETURN 鍵。輸入的資料完全依照緊跟在 GET 敍述後面的變數值的型式而定。打入以下的程式並執行它：

```
10   GET A$
20   PRINT A$
25   REM  IF ENTRY IS AN E, END PROGRAM
30   IF A$ = "E" THEN  END
40   GOTO 10
```

我們可以藉著測試的方法使 GET 敍述取到某一個特別的字：

```
10   GET A$
20   IF A$ <  > "X" THEN  GOTO 10
30   PRINT A$
40   END
```

這個程式除了等待輸入X字外，不做其他任何的工作。

如果 GET 敍述中使用的是整數或實數的變數，那麼輸入的資料必須是一個數字，否則的話您將得到 ? SYNTAX ERROR的錯誤訊息而程式也將停止執行；由於這個原因及其他的一些問題，使得GET

敍述通常都被使用在取得字串值。

當程式中需要與操作員對話的時候，GET 敍述最常被使用；例如，一個程式可能需要等待操作員證實他或她的存在——打入一個特殊的字（例如，Y）；以下就是這樣的一個例子：

```
10  PRINT "OPERATOR! ARE YOU THERE?"
15  PRINT "TYPE Y FOR YES"
20  GET A$
30  IF A$ < > "Y" THEN  GOTO 20
40  PRINT "OK, LET'S GET ON WITH IT"
50  END
```

需要注意的是，上面的這個程式不會將由鍵盤輸入的字顯示出來，您可以試著修改上面的程式使由 GET 敍述得到的字顯示在螢光幕上。

停止及繼續程式的執行

如果您想要停止正在執行的一個程式，那麼您只需同時按下 CTRL 鍵及C鍵即可，如果這個程式正在等待由鍵盤輸入INPUT敍述所需要的資料時，那麼您必須在按了CTRL - C 之後，再按下RETURN鍵。

在整數BASIC中，您在程式停止執行後將由螢光幕上看到STOPPED AT及程式停止在某一個行號的數目，您可以繼續這個被中途打斷的程式執行——打入 CON 命令。

在 APPLESOFT 中，程式被CTRL - C打斷後您可以由螢光幕上見到的訊息是 BREAK IN 加上程式停止處的行號，而使程式繼續的方法是您打入 CONT 命令。

RESET 鍵

當然您可以在任何時間按下RESET鍵使您的程式停止執行（在某些型式的APPLE Ⅱ上，您必須按下CTRL - RESET）。

在APPLE Ⅱ PLUS及具有語言系統卡的APPLE Ⅱ上，RESET與CTRL-C 具有同樣的功能。

在不具有自動監督系統的APPLE Ⅱ上，您按下RESET鍵會使得系統進入監督系統而您在螢光幕上將看到“ * ”的系統標示。如果您使用了整數BASIC 語言卡或是 APPLESOFT 語言卡，那麼按下CTRL-C將使您回到機器本身具有的語言系統內。如果您的APPLESOFT 系統是由磁帶內傳輸進入機器內的，那麼您可以打入 ∅G 使系統回到 APPLESOFT ，若是由磁片進入的，那麼您必須打入 3D∅G 回到 APPLESOFT 系統。

如果您想由不經意的RESET 回到您所在的系統中時，您在記憶體中的BASIC 程式將消失無蹤。

END敘述

如同我們在本章前面所描述的，當程式的執行碰到 END 敘述時，程式的執行就會停止。

在整數BASIC 中如果程式的執行碰到 END 敘述，那麼您就無法使它繼續執行下去了。

STOP 敘述

在 APPLESOFT 中有另外一個命令使得程式的執行停止，當APPLESOFT 執行STOP 命令以後，您將在螢光幕上看到BREAK IN 及程式停止所在的行號數目的訊息。

在 APPLESOFT 中您可以在程式碰到 STOP 或 END 敘述而停止後，打入 CONT 命令使程式繼續執行下去。

WAIT敍述

APPLESOFT 中有一個命令可以使程式在執行時暫時中止（PAUSE）；WAIT 敍述使得程式暫時中止執行直到您所指定的一個記憶體位置內存有某個您所指定的值爲止；例如：您可以使您的程式暫停，直到有人按下了遊戲控制器的第一個控制鈕爲止。下面就是這個程式：

```
10  REM  -WAIT FOR BUTTON ON GAME CONTROLLER NO.1
20  PRINT "PRESS BUTTON ON GAME CONTROLLER ONE"
30  WAIT  - 16286,128
40  PRINT "BANG!"
50  END
```

您可以參照第八章對WAIT的敍述做更進一步的瞭解。

BASIC語言提供的特殊函數

另一項BASIC語言的元素就是它所提供的一些特殊函數（FUNCTION），在某些方面說來，它似乎像是一個變數，但在另一方面它就如同一個BASIC敍述般的動作。

也許瞭解這些函數的最簡單方法就是看看一個例子，下面是一個指定敍述：

```
110 A=SQR(B)
```

變數A就是變數B的平方根，SQR就是表示開平方根的功能。下面是一個字串的特殊函數：

```
20 L=LEN(D$)
```

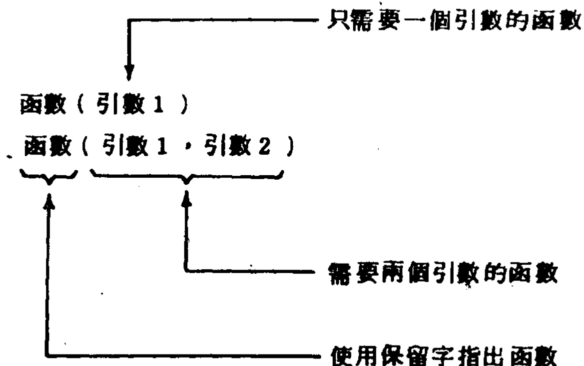
在這個例子中數値變數L就等於字串變數D$的長度。

特殊函數可以在等號的右邊代替BASIC敍述中的任何變數或是常數；換句話說，也就是您可以使用A= SQR(B)，而不能使用SQR(A)=B這樣的敍述。

以下的討論將告訴您如何去使用這些BASIC語言提供的特殊函數；許多在本章中未能詳述的函數將在第八章中做一個較詳盡的描述。有許多的函數在整數BASIC中並不存在，我們也將做一個記錄。

您使用保留字(RESERVE WORD)指出您想要的函數(例如，SQR就表示開平方根)，後面跟隨著用括弧括起來的引數(ARGUMENT)。像A=SQR(B)這樣的情形，SQR只需要一個引數，這個引數就是所要被取平方根值的值；至於L=LEN(D$)，LEN表示函數，而D$就是引數，也就是要被取長度的字串。

一般說來，任何函數都具有以下兩種格式的其中之一：



有一些函數需要三個引數；每一個函數的引數可以是數値、變數或是一個運算式。

每一個BASIC敍述中的函數在這個敍述的其他部份被計算之前都會被轉變成一個數値或是字串値。首先，函數的引數依照我們所定的規則計算出來，一旦引數被算出成為一個數値或是字串値之後，整個函數

就依照它的值而產生結果，並不是依照運算式計算的先後順序而依次計算函數的值。例如，以下的敍述：

```
110 B=24.7*(SQR(C)+5)-SIN(0.2+D)
```

SQR及**SIN**首先被計算；假設**SQR**(**C**)＝6.72，**SIN**(0.2+**D**)＝0.625，那麼這個運算式就變成：

24.7 * (6.72＋5)－0.625)

然後這個較簡單的敍述才被計算。

數值函數

以下是一些您可以在整數**BASIC**及**APPLESOFT**中使用的數值函數：

SGN	將引數的正負號產生出來；正值的引數則產生＋1，負值的引數則產生－1，若引數值爲0則產生0。
ABS	將引數的絕對值產生出來；對於正數而言，其值不變，對負值而言，則將其變爲正數。
RND	產生出亂數(**RANDOM NUMBER**)；詳細的情形在第八章中描述。

以下是一些您只能在**APPLESOFT**中使用的數值函數：

INT	將一個浮點(**FLOATING POINT**)的引數值轉變成相對的整數值。
SQR	求出引數的平方根。

EXP	求得 e 引數的值。
LOG	將引數的自然對數（NATURAL LOGARITHM）值求出。
SIN	將引數的三角函數正弦值求出。
COS	將引數的三角函數餘弦值求出。
TAN	將引數的三角函數的正切值求出。
ATN	將引數的三角函數的反正切值求出。

如何使用數值函數

您應該儘快地開始使用各種函數；但對您尚未瞭解的函數而言，您最好還是暫時不要去招惹它。例如，您如果對三角函數並不瞭解，那麼在您的程式中幾乎不可能使用到SIN、COS及TAN函數，而您學習它們的意義也並不十分的重要。

以下就是一個使用數值函數的例子：

```
10 A=-234
20 B= SGN (A)
30 PRINT B
40 END
```

當您執行這個程式時，程式所產生的結果就是－1，因爲－234是一個負值；您可以練習將行號 1∅ 的敍述改爲INPUT，行號 40 的敍述改爲GOTO 1∅，那麼您就可以輸入許多不同的A值而觀察 SGN 函數如何工作了。

下面是一個使用APPLESOFT 的數值函數的較複雜的程式：

```
10 INPUT A,B
20 IF  LOG (A) < 0 THEN A = 1 / A
30 PRINT  SQR (A) *  EXP (B)
39 REM  USE CTRL-C TO END PROGRAM
40 GOTO 10
```

如果您對對數（LOGARITHMS）有足夠的瞭解，那麼您可以練習將

行號 20 的敍述改掉，用其他的數值函數代替LOG函數。

字串函數

字串函數允許您在許多地方處理字串資料，您也許對不熟悉的數值函數不必太過瞭解，但對每一個字串函數而言，您必須多下一些功夫去研究它們。

下面是可以使用在整數BASIC及APPLESOFT中的一些字串函數：

ASC	將一個字轉變成相對的標準的數字碼（ASCII）。
LEN	將一個字串的長度求出來。

以下是一些只能在APPLESOFT中使用的字串函數：

STR$	將一個數值轉換成一個字串。
VAL	將一個字串轉換成它們相對應的數值（如果轉換是可以達成的話）。
CHR$	將一個數字碼（ASCII）轉換成相對的文字。
LEFT$	將一個字串的左邊部份取出，函數中第一、二個引數分別指出字串及所要取的左邊部份。
RIGHT$	將一個字串的右邊部份取出，函數中第一、二個引數指出字串及所要取的右邊部份。
MID$	將一個字串的中間部份取出；函數的第一、二、三個引數分別指出字串及所要取的中間部份。

字串函數使您可以決定字串的長度、取得字串的某一部份、轉換數值、數字碼（ASCII）及字串等等。這些函數需要一個、兩個或是叁個引數，以下就是一些例子：

STR$（14）	將14轉變成"14"。
LEN（"ABC"）	將這個字串的長度求得，求出的數就是3，因爲這個字串只有三個字。
LEN（A$+B$）	將A$及B$連在一起的長度求出。
LEFT$（ST$，1）	將ST$的最左邊一個字取出。

整數BASIC中的部份字串

雖然整數BASIC中沒有任何函數使您取得字串中的某一部份，但是有一個方法可以達成這個目的；您可以像下面這個例子一般地在部份字串（SUBSTRING）中指出所要取的字串的起始位置及所要取的字的數目：

```
10 DIM A$(20),B$(5)
20 B$=A$(1,4)
```

上面的例子中將把A$的起始四個字放入B$中；這個例子看起來似乎是將A$（）矩陣中的一個元素放入B$內，但是您應該記得在整數BASIC中並不允許字串矩陣的存在，更別說二階的字串矩陣了；因此上面的表示只是用在部份字串中，在括弧內的第一個值表示取字的位置從何處開始，第二個值則表示所要取的字的數目是多少。

整數BASIC中的字串連接

LEN 函數允許您在整數BASIC中將字串連接在一起。下面就是一個例子：

```
10  DIM A$(10),B$(10),C$(10)
20  A$="WIND"
30  B$="PIPE"
40  C$="LINE"
50  A$( LEN(A$)+1)=B$
60  PRINT A$
70  B$( LEN(B$)+1)=C$
80  PRINT B$
90  END

>RUN
WINDPIPE
PIPELINE
```

系統所提供的函數

我們在這裏也把系統提供的函數列出供您參考；不過在您成爲一個有經驗的程式設計師之前，您還不需要使用到它們。

以下就是一些系統所提供的函數：

PEEK	由某一個記憶體位置內取出它的資料。
FRE	將沒有用到的記憶體位置數目顯示出來——沒有被使用到的RAM的數元組數目；這在整數BASIC中是不可以用的。
USR	轉移到組合語言程式（ASSEMBLY LANGUAGE PROGRAM）；在整數BASIC中不可以使用它。

使用者自訂的函數

除了BASIC提供的函數之外，您在APPLESOFT中也可以自訂一些算術函數；使用者自訂的字串函數是不被允許的。DEF FN敍述定義了使用者自訂的函數，在您使用它時，就使用FN敍述即可。下

面是一個使用DEF FN敍述的小程式：

```
10  DEF  FN P(X) = 100 * X
20  INPUT A
30  PRINT A, FN P(A)
35  REM  USE CTRL-C TO END PROGRAM
40  GOTO 20
```

函數的辨識字（IDENTIFIER）跟隨在保留字FN的後面，至於定辨識字的規則與變數名稱的規則相同；在上面這個例子中我們使用P做爲辨識字，因此這個函數名稱就變成FNP，如果辨識字是AB，那麼函數名稱就是FNAB。

在等號右邊的算術運算式定義了這個函數所能從事的工作，當您將這函數用在FN敍述內時，那麼運算式就依照式中所有的值加以計算，而它的結果就像是一般的數值運算。

在DEF FN敍述中，函數辨識字之後必須跟隨著用括弧括起來的一個變數，而這變數名稱只對這個函數有意義，當它在DEF FN敍述之外時並沒有任何的意義，您可以在程式的其他地方使用這個變數名稱而不致影響到函數敍述所做的工作；而且在函數中的變數對程式中其他相似名稱的變數名稱也沒有任何的影響。

在使用的時候，FN敍述必須跟隨著函數辨識字，在函數辨識字之後必須跟隨著一個數值常數、變數或是運算式（要在括弧之內）；當APPLESOFT 由當遇到FN敍述的時候，它就會把常數的值、變數的值或是運算式的值放入在DEF FN敍述中的引數內（在DEF FN之外的同名稱的變數並不受到任何的影響）。出現在定義函數運算式內的變數，當APPLESOFT計算這個運算式時，只使用它當時被放入的值。

多重性的函數

函數的引數可以是一個運算式，而這個運算式中又可以包含著其它

的函數；換句話說，也就是函數也可以具有多重性。以下就是一個例子:

```
10   INPUT A
20   PRINT  SGN ( ABS (A))
25   REM  USE CTRL-C TO END THE PROGRAM
30   GOTO 10
40   END
```

試著在立即式的情況下將數值函數與字串函數放在PRINT敍述內使用。

4

深入地瞭解BASIC 程式語言

這一章是第三章的延續，在本章中我們將告訴您如何地使用APPLE Ⅱ BASIC 語言設計程式，並將許多新的BASIC敍述展示出來，同時將您已往熟悉的BASIC敍述作更深一層的分析。在第三章中，我們已經使您能夠使用APPLE Ⅱ做一些普通的工作，這一章中將繼續使您更加深入的瞭解。

直接地使用及控制APPLE Ⅱ

有許多的敍述可以使您直接地使用APPLE Ⅱ，例如，您可以用這些敍述來控制遊戲、操作發聲器及完全地使用外圍的各種設備。

記憶體及位置

APPLE Ⅱ可以擁有65,536個記憶體位置(LOCATION)，每一個記憶體位置可以儲存一個在Ø與255之間的數字（這個奇怪的上限實

際上就是 2^8 ），所有的程式及資料都被轉換成一系列的數字儲存在記憶體內。

在以下的BASIC敍述中，若您必須指出一個記憶體位置，您可以使用一個數字、變數或運算式來做這個工作，但是不論您使用的是那一種方式，它所代表的必須是一個有效的記憶體位置。事實上，記憶體位置有兩種不同的表示方法，一種是處於 ∅ 與 65535 之間的正整數；另外一種則是一個負值，它可以由正值的記憶體位置減去65536而得到。例如，－32768與 32768 表示同一個記憶體位置，而這個位置及可以用－1或65535來表示的記憶體位置是兩個在整數BASIC中所無法直接使用到的記憶體位置；當您記起在整數BASIC中所允許的最大數目是32767時，您就可以發現使用負值來代表記憶體位置的功用了。如果您在 APPLESOFT 中使用一個實數值來代表記憶體位置，那麼APPLESOFT將先把它轉換成整數值。

PEEK與POKE

PEEK 的功能就是使您能夠讀到存在APPLE Ⅱ 記憶體內任何位置中的值。考慮以下的敍述：

```
10 A = PEEK(200)
```

這個敍述把存在記憶體位置200 的值放入變數A內。

POKE 敍述則把一個值放入到一個記憶體位置去。例如，下面這個敍述：

```
20 POKE 8000,A
```

把變數A的值放到記憶體位置 8000 內，而被寫入記憶體內的值可以是一個數字、變數或是一個運算式所產生的值，但是它的值一定得在 ∅ 與 255 之間。

您可以使用 PEEK 敍述查驗 RAM或ROM 中的內含，但是您只能使用 POKE 把值存放到 RAM 內；這就如同ROM（READ ONLY MEMORY）的名字所顯示的，它只能被讀而不能寫入任何東西。

CALL敍述

您可以由BASIC程式把控制轉移到一個組合語言程式（ASSEMBLY LANGUAGE PROGRAM）或到一個組合語言的副程式去（使用 CALL 指令）；看看下面 CALL 敍述的例子：

```
100 CALL A1
```

控制轉移到變數 A1 內所存的記憶體位置的程式去了。

這個組合語言的程式或副程式可以是存在 ROM 中的或是您自己設計的；例如，監督程式就具有一個固有的副程式，它可以清除整個顯示螢光幕，在附錄D中有一個詳細表列，將您所能使用到的APPLE Ⅱ本身具有的副程式詳列出來；同時，您可以參考第七章以便瞭解監督程式及組合語言。

HIMEM：及LOMEM：敍述

APPLE Ⅱ 的記憶體用在許多不同方面，好比說，一部份被用來儲存您的BASIC程式，一部份儲存您程式中的變數，另外一部份則用來處理磁碟機系統（如果您使用了磁碟機）。又有一些的記憶體被用來處理圖形，這將在第六章中爲您陳述。

HIMEM：及LOMEM：敍述使您能夠設定好記憶體位置，使得不致於被組合語言程式及高解析度的圖形（HIGH-RESOLUTION GRAPHICS）所干擾。

以下就是這兩個敍述的一個例子：

```
50 HIMEM: 38400
60 LOMEM: 12291
```

這是在您的APPLE Ⅱ中，機器自動地為您的BASIC程式所設定的最高及最低的記憶體位置，如果您需要使用高解析度的圖形或組合語言程式，那麼您也許必須重新設定這個上下限，這在第六及第七章將詳細地告訴您。至於記憶體的如何使用等資料，您可以參考附錄G。

使用週邊設備

當您將APPLE Ⅱ打開後，它就期待著您由鍵盤輸入資料並把資料由顯示螢光幕上顯示出來；鍵盤就是標準的輸入設備而顯示螢光幕就是標準的輸出設備；但就如同您瞭解的，APPLE Ⅱ能夠與外在的各種輸入及輸出設備（PERIPHERAL INPUT AND OUTPUT DEVICES）作聯繫，而這些設備包括：

- 一個可以儲存程式及資料的磁帶機。
- 可供儲存程式及資料的磁碟機。
- 一個可以將資料或程式複製在紙上的列表機。
- 一個可以用手操作自由繪圖的繪圖板。
- 與其他的電腦作各式的聯繫。

磁帶機與APPLE Ⅱ經由APPLE Ⅱ後面的兩個特殊的接點連接。至於其他的輸入及輸出設備全都與插在APPLE Ⅱ的擴充接點上的電路板連接，而這些電路板就被稱為控制卡（CONTROLLERS）、介面卡（INTERFACE CARDS），或是就說成卡（CARDS）、介面（INTERFACES）。您在使用某一個設備時只需指出它所連接的介面卡所在的擴充接點數目即可；APPLE Ⅱ具有八個擴充接點——它們的編號分別由ø到7；至於顯示螢光幕及鍵盤則永遠是擴充接

點∅。

PR# 及IN# 敍述

我們使用 PR# 敍述來選擇一個輸出的擴充接點（ SLOT ），當選擇輸入擴充接點時，我們使用 IN# 敍述。下面的例子就是選擇顯示螢光幕作爲輸出的工具：

```
PR#0
```

只要磁碟作業系統沒有放在機器內，那麼您在立即式（或間接式）中使用PR# 與 IN# 是沒有差別的；至於當DOS 已經在機器內時，在間接式的情況下使用 PR# 與 IN# 就稍爲複雜一些了；您必須使用PRINT敍述印出CTRL-D這個字，並在後面緊跟著PR# 或 IN# 命令；下面的敍述使得PRINT敍述的訊號輸出到擴充接點 1 上，然後藉著插在擴充接點 1 上的介面卡，把輸出的資料輸送到列表機或是任何接在這個介面卡上的裝置去。

```
100 D$ = "": REM  CTRL-D
110  PRINT D$;"PR#1": REM  SELECT SLOT NUMBER 1 (PRINTER)
```

但是如果 DOS 並不存在機器內，那麼您只需使用下面的敍述代替就可以了：

```
110 PR#1 : REM SELECT SLOT 1 (PRINTER)
```

需要注意的是，PR# 及 IN# 各自具有一個元素，而元素的值必須是在∅到 7 之間的正整數，使用其他任何的值可能會發生不可預測的後果；如果您使用 PR# 或 IN# 所選擇的擴充接點上沒有介面卡，那麼APPLE Ⅱ 將被鎖住（LOCK UP），這種情形的唯一解法就是您按下 RESET 鍵。

程式的輸出及資料的輸入

即使最沒有經驗的程式設計師也能很快地發現一個程式的輸出及輸入部份是它最能變把戲的地方。

幾乎每一個程式都是由鍵盤輸入資料的，那麼使用許多的INPUT敍述就足夠了嗎？在許多的情形下答案是否定的。如果操作員按錯了鍵，或者他或她在輸入了幾筆資料後才發現了錯誤，那麼是否產生了許多的麻煩呢？至於一個有用的程式它必須瞭解操作員是人，而人必定會犯一些錯誤的。

同樣地，程式所產生的結果並不能只靠執行一些PRINT敍述將它們顯示或印出即可；人必須去"讀"這個結果；除非輸出的資料經過小心地設計，否則它將是很難瞭解的，而往往會造成許多的錯誤。

幸運的是，APPLE Ⅱ的BASIC具有許多的功能使得您能正確地設計輸出及輸入的型式；我們在練習如何從事一個好的輸入與輸出工作前，先對這些功能做一個描述。

PRINT敍述的更深一層認識

正常的情況下，PRINT敍述結束後將把游標移到下一個顯示行的起始處去，這使得下一個PRINT敍述將所要顯示出的資料由下一列的第一個位置開始；因此下面的這個程式把 20 個W字分別顯示在每一個顯示行的起始位置上：

```
>NEW

>200 C$="W"
>210 FOR I=1 TO 20
>220 PRINT C$
>230 NEXT I
```

```
>240 PRINT "PHEW!"
>250 END
>RUN
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
W
PHEW!
```

如何使用分號

跟在PRINT敍述內任何變數後面的分號（SEMICOLON）使得下一個變數的顯示位置緊跟在這個變數的後面。至於跟隨在PRINT敍述的最後一個變數後面的分號則使得游標回頭的功能消失。因此，下面的這個程式將把800個W字連續地顯示在螢光幕上（使用了 20 列，每列 40 個字的螢幕行）。

```
>NEW

>200 C$="W"
>210 FOR I=1 TO 800
>220 PRINT C$;
>230 NEXT I
>240 PRINT "PHEW!"
>250 END
```

```
>RUN
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
PHEW!
```

在**FOR-NEXT**裏面的脚註變數I被使用做爲一個計數器，指出被顯示出的W字的數目，在這個例子中I就等於800。當**PRINT**敍述開始被執行時，一個新的顯示行開始而第一個W字就被顯示出來了，這時由於分號的存在停止了游標回頭功能的執行，因此游標仍然位於第一個W字的後面，而第二個W字被顯示出來以後，游標就往右邊移動了一位，這個順序直到第一行都被佔滿了爲止，這時游標就往下一行移動了；而這整個的順序一直繼續直到20行都被佔滿爲止（每個顯示行顯示40個字）。

那麼爲什麼**PHEW**！顯示在一個新的顯示行呢？事實上並非如此，這個字顯示在一個新行的原因是因爲最後一個的W字佔用了顯示行的最後一個位置；如果您把重複執行的數目由800減到780，那麼**PH-EW**！就會顯示在最後一行的結尾處了。這種情形可以由下面的說明解釋：

```
>200 C$="W"
>210 FOR I=1 TO 780
>220 PRINT C$;
>230 NEXT I
>240 PRINT "PHEW!"
>250 END

>RUN
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWWW
WWWWWWWWWWWWWWWWWWWWPHEW!
```

分號的功用就是把字串資料及數值資料的顯示連接在一起而中間不留任何空隙。

爲說明這一點，您可以把上面的程式中的字串資料改爲數值資料，而這程式就會如同下面的一般：

```
>200 C=5
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "PHEW!"
>250 END
```

數字 5 將顯示成下面的情形：

```
>RUN
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
```

```
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
5555555555555555555555555555555555555555
PHEW!
>
```

現在把C的値改爲－1：

```
>200 C=-1
>210 FOR I=1 TO 800
>220 PRINT C;
>230 NEXT I
>240 PRINT "PHEW!"
>250 END

>RUN
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
```

```
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1
PHEW!
```

您注意到什麼情形發生了嗎？當C＝－1時，這程式顯示出來的顯示行的數目是當C＝5時的一倍，因爲它必須使用一個額外的位置把負號顯示出來。

具有許多位數的數字當顯示時（使用上面的程式），將使螢光幕的顯示往上捲而把先前的顯示隱藏起來，除非您將FOR-NEXT的脚註加以調整。如果C的值是2001，那麼它就需要四個位置來顯示，因此將脚註調整爲　800/4＝200　：

```
>200 C=2001
>210 FOR I=1 TO 200
>220 PRINT C;
>230 NEXT I
>240 PRINT "PHEW!"
>250 END
>RUN
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
 20012001200120012001200120012001200120012001
```

```
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
2001200120012001200120012001200120012001
PHEW!
```

由於分號使得顯示的情形連接在一起（除非達到一行結束的地方），因此數字的顯示可能被分開在兩行顯示。您可以試著使C = 201而執行上面的例子看看結果如何？

使用逗號

在PRINT敍述中的變數後面或PRINT敍述結束的地方使用逗號，使得在顯示資料時螢光幕上將自動地預留空白的位置。這種預留空白位置的規則在整數BASIC與APPLESOFT 中稍有不同。

在整數BASIC中螢光幕的顯示設定爲五個停止位置，它們分別是1、9、17、25及 33 ，每兩個位置之間具有八個空白位置。以下就是一個整數BASIC的例子：

```
>NEW
> 200 C=123
> 210 FOR I=1 TO 100
> 220 PRINT C,
> 230 NEXT I
> 240 PRINT "PHEW!"
> 250 END
>RUN
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
123     123     123     123     123
```

```
123        123        123        123        123
123        123        123        123        123
123        123        123        123        123
123        123        123        123        123
123        123        123        123        123
123        123        123        123        123
123        123        123        123        123
PHEW!
```

在每一個停留位置之前必須有一個空白的位置（除了第一個停留位置之外），否則這個停留位置將不生效；因此對第二個停留位置而言，想要使它發生作用就必須在第八個位置預留空白；同樣地，對其他的停留位置而言，也必須不使用到它們前面的那一個位置。下面的例子就是使用字串來說明這種情形。

```
>10 REM MUST DIMENSION STRING VARIABLES
>20 DIM C$(8)
>200 C$="12345678"
>210 FOR I=1 TO 20
>220 PRINT C$,
>230 NEXT I
>240 PRINT "PHEW!"
>250 END
RUN
12345678                12345678                12345678
                12345678                12345678
12345678                12345678                12345678
                12345678                12345678
12345678                12345678                12345678
                12345678                12345678
12345678                12345678                12345678
                12345678                12345678
PHEW!
```

如果您把 C$ 改為“1234567”，那麼所有的五個停留位置都將發生作用。

在 APPLESOFT 中使用逗號控制停留位置必須遵照一些特別的規定，在圖 4－1 中我們有一個說明。

我們可以使用前述的例子把分號改為逗號看看結果如何；這個程式將使得數字顯示在三個行（COLUMNS）上，而每個顯示行擁有三個數字，因此 FOR-NEXT 的腳註必須是 3＊20＝60，而這程式就如同下面所顯示的一般。

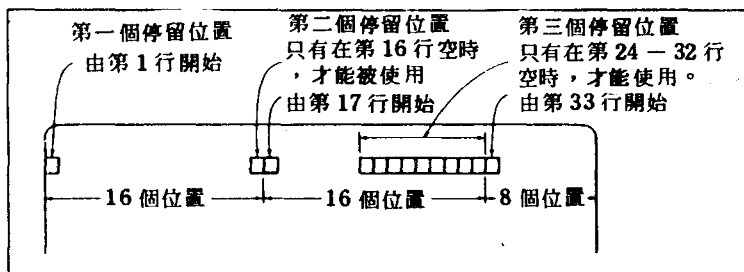


圖 4 — 1 在 APPLESOFT中 PRINT敍述使用逗號螢幕上停留的位置

```
]200 C = 2001
]210 FOR I = 1 TO 60
]220 PRINT C,
]230 NEXT I
]240 PRINT "PHEW!"
]250 END
RUN
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
2001          2001          2001
PHEW!
```

逗號在字串的顯示中依然發生作用；例如，輸入下面的程式將 20 個顯示行的字串顯示出來（每個顯示行包含三個資料）。

```
]NEW

]100 A$="THREE"

]110 B$="BLIND"

]120 C$="MICE"

]210 FOR I=1 TO 20

]220 PRINT A$,B$,C$

]230 NEXT I

]240 PRINT "PHEW!"

]250 END

]RUN
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
THREE           BLIND           MICE
PHEW!
```

我們使用**VISIONARY**代替**BLIND**放入 B$ 做爲更進一步的練習；如果您研究過圖 4－1，那麼您將對爲何第三個停留位置沒有發生作用毫不訝異；因爲**VISIONARY**這個字，當被顯示在第二個停留位置時，它佔用了由 24 到 32 的位置，而顯示在 24 到 32 的位置內的任何資料將使得第三個停留位置不發生作用。

顯示格式的一些功能

我們使用格式化（FORMATTING）這個名詞描述將訊息安排成易於瞭解的型式在螢光幕上顯示的這個動作。如同我們前面所介紹的，逗號及分號都具有這方面的功能；當然，格式化可能是一件複雜的工作；例如，假設您想要把一個標題放在螢幕上端一行的中間，那就可能需要花費許多工夫了。

下面是一個整數BASIC的程式，它使用逗號將標題印在螢光幕的中間位置，同時這個程式使用了 CALL 敍述，這個敍述使得游標位於螢光幕的左上角處（參看附錄D）。

```
>NEW

>5 REM MUST DIMENSION STRING VARIABLES
>10 DIM HD$(30)
>500 REM CLEAR THE SCREEN AND
>510 REM MOVE THE CURSOR TO THE
>520 REM UPPER LEFT CORNER
>530 CALL -936
>540 HD$="TOP SECRET"
>550 REM TAB TO CENTER OF SCREEN
>555 REM REM THEN PRINT HEADING
>560 PRINT "","",HD$
>570 END
>RUN
                    TOP SECRET
```

這個標題大約在螢光幕的中央，但是如果我們使用一個較長的標題，如REPORT ON HEAVY HYDROGEN，那麼就不可能使用上面的程式將它放在中央的位置了；如果您把上面的程式修改後，這個標題是可以使它位於大約中央的位置——取消一個逗號及與它連接的虛字串（NULL STRING），使標題由第二個停留位置開始顯示。

我們可以在APPLESOFT 中使用同樣的技巧顯示出標題，但是APPLESOFT 中有許多使PRINT敍述格式化的工具——如 SPC TAB 及POS 等功能。

SPC功能

SPC 是一個跳過空白（SPACE-OVER）位置的功能。您把 SPC 功能包含在PRINT敍述以內，就可以執行跳過空白的功能；在 SPC 這個字之後必須跟隨著用括弧括起來的數字（用以表示您希望跳過的空白位置的數目）。例如，我們可以使用下面的程式將 TOP SECRET 這個標題放在螢光幕的中央位置：

```
]100 REM CLEAR SCREEN
]110 HOME
]120 REM SPACE OVER AND DISPLAY
]130 PRINT SPC(15);"TOP SECRET"
]140 END
]RUN
               TOP SECRET
```

注意跟在 SPC 後面的分號。跟在 SPC 後面的逗號將使得您想要印出的資料印在空了 SPC 所指出的位置以後的下一個停留位置上，其他的都不改變。

TAB功能

TAB 如同打字機上的預定位置（TABBING）的功能一般。

假設您想要在某一列上印出或顯示訊息，您必須首先計算它由那一個行開始的位置，而在附錄L的格式就是爲了這個目的而設計的；在圖4－2中，行的開始位置分別是1、15及31。現在您可以使用TAB功能代替空白的計算——在PRINT敍述中資料的後面放入TAB功能。

考慮圖4－2中的一個顯示行，您可以用計算位置的方法把這一個

```
1  行 號                        15                              31
JONES.P.J                    431-25-6277                     1420.00
BURKE.P.L                    447-71-7614                     2025.00
ROBINSON.L.W                 231-80-8421                     2150.00
         etc.                     etc.                          etc.
```

圖4－2　決定螢光幕上字的位置

顯示行顯示出來（不使用停留位置的功能）：

```
]10 ?"JONES,P.J         431-25-6277       1420.00"
```

我們可以使用 SPC 功能代替空白位置的加入而把程式縮短成下面的情形：

```
]10 ?"JONES,P.J";SPC(7);"431-25-6277";SP
C(5);"1420.00"
```

但是使用 TAB 功能是更為容易的了，因為它使您直接預定您所知道的位置而不必去計算空白了。

```
]10 ?"JONES,P.J";TAB(16);"431-25-6277";T
AB(32);"1420.00"
```

決定游標的水平位置

POS 是 **APPLESOFT** 中最後一個格式化的功能，**POS** 告訴您游標位於那一行的位置，而這位置用一個數字來表示，就代表著在閃爍的游標的位置。在 **POS** 後面您必須使用一個虛元素 ø，如，**POS**（ø）。

下面的敍述對**POS** 的功能做了一個示範：

```
]?"CURSOR POSITION IS";POS(0)
```

在立即式情況下執行這個敍述，您將從顯示螢光幕上看到：

```
]?"CURSOR POSITION IS ";POS(0)
CURSOR POSITION IS 19
```

在將 **CURSOR POSITION IS** 顯示出來後，游標位於第 19 個位置上；如果您在 **I S** 後面而在右引號之前加入空白位置，那麼您將得到較 19 更大些的數字。

在整數**BASIC**中您可以引用**PEEK(36)** 模擬**POS(∅)** 的功能。下面是一個前面例子的改寫：

```
>PRINT "CURSOR POSITION IS ";PEEK(36)
CURSOR POSITION IS 19
```

需要注意的是，顯示螢光幕的行數是由 ∅ 到 39 （不論您使用**POS**、**PEEK (36)** 及**PRINT SPC**）；至於**PRINT TAB**，行數則由 1 到 40，而**HTAB** 及**TAB**（整數**BASIC**）也是由 1 到 40。

決定游標的垂直位置

您可以使用**PEEK (37)** 而得到游標所在列的位置。

列的編號在**PEEK (37)** 時是由 ∅ 到 23，而在**VTAB** 中則是由 1 到 24。

游標的控制及螢幕顯示的特殊效果

有許多的**BASIC**敍述增加了**APPLE Ⅱ**螢光幕的顯示功能，它

們包括了FLASH、INVERSE、SPEED、NORMAL、HOME、VTAB及HTAB，TAB敍述，它們大多數只能在APPLESOFT中使用。還有許多的圖型顯示敍述，我們將在第六章中描述它們。

定游標的位置

我們已經描述了在PRINT敍述內使用逗號及分號來控制游標的位置，而APPLESOFT中的SPC、TAB及POS等功能也對定游標的位置非常有用。

清除顯示螢光幕

您可以藉著執行CALL－936這個敍述或使用HOME敍述（在APPLESOFT中）將螢幕消除並將游標放到母位的位置（左上角）。試著打入以下的敍述（前面的用在整數BASIC中，後面的則用在APPLESOFT中）：

```
>CALL -936
]HOME
```

平行及垂直地定出游標位置

有兩個敍述使得您可以把游標移到螢幕的任何位置去；VTAB垂直地移動游標而HTAB（在整數BASIC中用TAB）平行地移動游標。在VTAB中您必須指出顯示行數（LINE NO），而在HTAB中您必須指出行數（COLUMN NO）；顯示螢光幕的最上方顯示行數目是1而最下方的顯示行數目則是24；至於最左邊的行數是1而最右邊的行數就是40。

以下的APPLESOFT的程式使用這兩個敍述在將游標定了位置之後顯示出一個星號；如果您使用整數BASIC語言，那麼您必須在行號90及120的敍述中用TAB代替HTAB。

```
90   HTAB 1: VTAB (1)
100   INPUT "ROW?";R
110   INPUT "COLUMN?";C
120   VTAB R: HTAB C
130   PRINT "*";
140   GOTO 90
```

INVERSE及NORMAL敍述

您可以使用INVERSE敍述將顯示在螢光幕上的字以白底黑字顯示出來。只要INVERSE敍述被執行，任何已經由PRINT敍述所顯示的資料都會變成白底黑字；但是，您由鍵盤所輸入的資料將依然照正常的情形顯示。

當APPLE Ⅱ執行了NORMAL命令之後，顯示的情形就會回到正常的情況下。

您可以試著執行下面的敍述，看看它們如何工作（有陰影的字就表示是白底黑字的）：

```
]INVERSE

]?"BLACK ON WHITE"
BLACK ON WHITE

]NORMAL

]?"WHITE ON BLACK"
WHITE ON BLACK
```

在整數BASIC中，INVERSE及NORMAL敍述是不存在的。

FLASH敍述

不僅僅您可以調換字的黑白色顯示情形，您也可以使用 **FLASH** 敍述使字的顯示處於正常與相反的兩種情況下交替不斷地閃動；同樣地，我們使用**NORMAL**敍述使**APPLE Ⅱ**回到正常的顯示情況下。

以下就是一個例子（有陰影的地方表示閃動）：

```
]FLASH
]?"FLASH IN THE PAN"
FLASH IN THE PAN

]NORMAL

]?"STEADY AS RAIN"
STEADY AS RAIN
```

FLASH 在整數**BASIC**中並不存在。

SPEED敍述

字在螢光幕上顯示的速度是一個變數，您可以使用 **SPEED** 敍述減慢它的顯示速度。

下面的程式說明 **SPEED** 敍述的工作情形：

```
]100  INPUT "SPEED = ";SP
]110  SPEED = SP
]120  FOR CT = 1 TO 3
]130  PRINT "HIC"
]140  NEXT
]150  PRINT "HICCUP"
]160  SPEED = 255
]170  END
```

在上面的例子中 **SP** 的值調整顯示的速度：∅ 是最慢的而255是最快的速度。**SPEED** 同時也會影響到字往其他設備輸送的速度，不僅僅是螢光幕而已。

在整數**BASIC**中，我們不能使用 **SPEED** 敍述。

顯示幕的文字資料幕

APPLE Ⅱ 在正常的情況下，使用 24 顯示行（每行 40 個字）來顯示所有字的資料。您可以使用 **POKE** 敍述改變文字資料幕（**TEXT WINDOW**）的大小及位置；在表 4 － 1 中列出了在記憶體中控制文字資料幕大小、位置等的記憶位置。

您在設定文字資料幕的時候，必須依照表 4 － 1 的限制從事，如果您設定的範圍超出了表 4 － 1 所能容許的限制，那麼可能發生不可預測的後果。

下面的例子在顯示幕的中間設定了一個兩行的文字資料幕以便輸出一個數值；您可以藉著輸入非數值的資料並觀察會產生什麼樣的錯誤訊息而瞭解這一項技巧。需要注意的是，設定文字資料幕並不會把螢幕的其他部份消除掉，也不會將游標移到文字資料幕以內，您必須使用其他的**BASIC**敍述達到這些目的。

表 4 － 1　文字資料幕的記憶體位置

記憶體位置	控　　制	准許範圍
32	左邊的極限	0到39
33	寬　度	可在1到40之間但必須減去左邊極限值
34	最高行的位置	由∅到最下方行的數目
35	最下方行的位置	最高行減去24

```
1000  REM  SET WINDOW TOP LINE, WIDTH, LEFT MARG., & BOTTOM
 LINE
1010 T = 10:W = 20:LM = 11:B = 13
1020  REM  CLEAR SCREEN
1030  CALL  - 936
1040  REM  SET TEXT WINDOW FOR INPUT
1050  GOSUB 3200
1060  REM  SURROUND INPUT WINDOW WITH ASTERISKS
1070  GOSUB 3000
1080  REM  MOVE CURSOR INSIDE WINDOW; INPUT
1090  VTAB T + 2
1100  INPUT M1
1110  REM  RESET WINDOW TO FULL SCREEN
1120  GOSUB 3300
1130  REM   MOVE CURSOR TO BOTTOM LINE
1140  VTAB 23
1150  END
2990  REM  SURROUND WINDOW WITH ASTERISKS
3000  VTAB T + 1
3010  GOSUB 3100
3020  VTAB B + 1
3030  GOSUB 3100
3040  RETURN
3090  REM  PRINT ASTERISKS
3100  FOR I = 1 TO W
3110  PRINT "*";
3120  NEXT I
3130  RETURN
3190  REM  SET INPUT TEXT WINDOW
3200  POKE 32,T
3210  POKE 33,W
3220  POKE 34,LM
3230  POKE 35,B
3240  RETURN
3290  REM  SET FULL-SCREEN WINDOW
3300  POKE 32,0
3310  POKE 33,40
3320  POKE 34,0
3330  POKE 35,24
3340  RETURN
```

CHR$功能：如何使用ASCⅡ字

在上一章中我們曾經介紹了如何將您看不到的字產生出來，例如，CTRL-G使得APPLE Ⅱ 發出"嗶"的聲音，但是仍然有許多的字(包含了看不到及看得到的字)您無法由鍵盤直接產生，而這些字包括了"["及"＼"。您可以由APPLESOFT的CHR$功能中產生這些字。

在瞭解 CHR$ 功能之前，您必須先對字是如何儲存在APPLE Ⅱ的記憶體中這件事要有認識；這實在是非常簡單，因為電腦的記憶體只能儲存數字，而不能儲存文字，所以文字都是被轉換成數字碼（NUMERIC CODE）；APPLE Ⅱ與其他的微電腦都是使用ASC Ⅱ（AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE）碼做為標準。例如，在ASC Ⅱ碼中，使用 65 代表A字， 66 代表B字， 67 代表C字；您可以在附錄 I 中看到完全的ASC Ⅱ碼的對照表。無論何時，只要APPLE Ⅱ使用了字串，那麼它就會將文字變為ASC Ⅱ碼的數值。

在 APPLESOFT 中，如果您無法在鍵盤上將一個文字輸入到字串內，您仍然可以使用ASC Ⅱ碼將這個字找出並且使用。

CHR$ 功能就是將ASC Ⅱ碼轉換成相對應的字。例如，想要建立"$"這個符號，那麼首先在附錄 I 中找到它的相對應的ASC Ⅱ碼，然後就如同下面般的使用 CHR$ 功能：

```
]PRINT CHR$(36)
```

您可以採用立即式的方式把 0 到255 的ASC Ⅱ碼在 CHR$ 函數內使用。

您也可以在PRINT敘述內，使用 CHR$ 與正常的字串連接在一起：

```
]?CHR$(34);"ZOUNDS!";CHR$(34)
"ZOUNDS!"
```

CHR$ 函數使得您能夠將無法正常使用的字加在字串內；例如，游標回返及引號等等。

如何用程式輸入資料

任何程式的目標應該是如何將資料的輸入工作減到最少，並且在錯誤發生的時候如何使操作員很容易地修正它們，當然有許多的方法來組織輸入的資料以使減少發生錯誤的機會到最小。

首先，我們可以輸入一整組的資料然後去處理它們；當然，如果我們對於每一筆輸入的資料都立即地去處理，那麼這個程式必定不是一個好的程式。

例如，對於一個處理信件資料的程式，它必定需要輸入的資料就是名字及地址，那麼您就應該將名字及地址當做一組相關的資料；換句話說，也就是您的程式必須詢問名字及地址，使得操作員能夠同時輸入它們的資料，然後在有需要的時候能夠修改它們之中的任何部份。當操作員認為輸入的名字及地址都是正確的以後，程式才繼續地執行下去，這時程式應當詢問下一個名字及地址。

如果您的程式首先詢問名字，並處理它的資料，然後再詢問地址，再分別處理；將名字與地址當做不同的單位，那麼您的程式設計就需要多多的練習了。

對於輸入的資料暫時地留在螢光幕上這樣的組織方法不失為一個好的主意，這使得操作員能夠有機會發現錯誤並修正它們；如果輸入的資料在被輸入後馬上就在螢光幕上消失，那麼操作員就根本沒有發現錯誤的機會了；當然，不論您的程式如何設計，都必須使操作員有方法能夠修正輸入資料的錯誤。

在某些情況下，將相關的資料當做一組的資料輸入，然後可以修正輸入錯誤的方式並不是最好的輸入方法，而這種情形時常是令人吃驚的，它發生的情況經常是在想要由鍵盤輸入許多筆資料的時候；例如，假設一個鍵盤操作員必須在一天內輸入幾百筆的名字及地址的資料，而這資料輸入的程式不理會任何輸入資料的錯誤情況——即使操作員立即就

發現了這個錯誤；我們憑經驗的累積，發現錯誤最少的資料輸入是在當操作員發現錯誤的同時就能修正它們。而在這種情形時，操作員必須不理會錯誤的資料同時儘快地輸入資料，在這種時候，資料往往被輸入兩次（最好由不同的人），而由另一個程式比較兩次輸入的資料；這麼做的原因是因爲兩個不同的操作員同樣地輸入一筆錯誤資料的機會非常小，您可以信任由比較程式的比較結果可以找出錯誤的資料；當然，隨後必須有一個修正錯誤的程式做錯誤的修正工作。

問答式的資料輸入方法

一個具有問答式資料輸入方法的程式，利用顯示出來的指令及提示訊息指引使用者；我們使用一個非常簡單的程式對這種方法做一個示範，我們將改變前面所介紹過的一個程式使得它具有問答式資料輸入的功能，而將這個程式變得更簡單。

以下就是前面介紹過的程式：

```
200 C$ = "W"
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "PHEW!"
250  END
```

如同前面介紹的，這個程式將印出800個W字，後面再印出PHEW這個字，這個程式在整數BASIC與APPLESOFT中同樣地可以被執行。

假設我們想印出不是W的字。

首先我們將程式中的設定敍述取消；您應該記得想要取消一個程式敍述的方法，就是打入這一個敍述的行號並緊接著按下RETURN鍵。

```
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "PHEW!"
250  END
```

這時行號 200 的敍述已經不存在了。

這時使用立即式的命令打入C$="X"的敍述，並執行這個程式。

```
]C$="X"

]RUN
PHEW!
```

這個程式只印出了 **PHEW**！而沒有顯示出X字；顯然地C$的值並沒有被輸送到程式內去。

RUN 這個命令使得程式在執行之前把所有的數值變數消爲 ø，並把所有的字串變數消除爲虛無字串（ **NULL** ）。所以C$裏面就被清爲虛無字串了，而一個虛無的字串就在程式的迴圈內顯示出來了（虛無也就是沒有任何東西，它無法顯示出來，也無法使游標移動）。

那麼是否有方法可以在每次執行程式的時候輸入不同的值到C$裏面去呢？您可以使用**GOTO** 210代替**RUN**，來達成這個目的，因爲**GOTO** 命令並不改變任何變數的值。

```
]C$="X"

]GOTO 210
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
PHEW!
```

使用上面的方法執行這個程式只有兩個步驟，但是它的過程是笨拙的；因爲您必須打入一個設定敍述，而且如果您使用 RUN 代替GOTO時，您必須重新來過。由於程式可以在執行時經由 INPUT 敍述得到顯示幕上的資料，您可以打入下面的程式行：

```
]200 INPUT C$
```

現在您可以在顯示幕上列印出這個程式並確定您是否正確地輸入了這一個程式行。

```
]LIST

200   INPUT C$
210   FOR I = 1 TO 800
220   PRINT C$;
230   NEXT I
240   PRINT "PHEW!
250   END
```

當您執行這個程式的時候，游標就顯示在下一行的位置（在APPLESOFT中，您也可以看到在游標的旁邊有一個問號），這時只要您打入任何一個字，這個字就會被重複地在螢光幕上顯示 800 次；您若再重新執行這個程式，並且重新輸入另外一個字，那麼顯示幕上顯示的就是這個新被輸入的字；需要注意的是，如果您在整數BASIC中一次輸入不只一個的字，那麼程式的執行將會停止，並且在顯示幕上顯示出 *** STR OVFL ERR 的訊息，這是因爲 C$ 沒有被宣告爲一個矩陣的關係。

上面的改變對原先的程式來講是一個改進的措施，但是，只是在顯示幕上看到游標在那裏閃動，並不能告訴操作員做什麼動作，似乎還不夠；因此您可以加入一行提示訊息，要求操作員輸入一個字。輸入下面的程式行：

```
]190 PRINT "ENTER ONE CHARACTER"
```

將這個程式列印出來同時檢查您的輸入是否正確。

現在這個程式已經具有說明操作的能力了，您再重複地執行這個程式，就可以發現這個程式是多麼容易地被使用了。

現在只剩下一個重要的修改工作，如果您想要這個程式自動地被重複執行，您就必須使用一個 **GOTO** 敍述將程式的執行轉移到程式的起始處，而不使用結束執行的敍述。那麼您就不需要打入 **RUN** 來重新執行這個程式了。加入以下的程式行：

```
1250 GOTO 190
```

現在這個程式是更容易被使用了，您只需要輸入 **RUN** 指令，然後就只需要遵守它的指示就可以了。

當然，現在您必須使用 **C** TRL-**C** 來結束這個程式；但是您也可以設計一個特別的檢查方法，使得在輸入某一個字的時候，程式自動地停止執行。例如，您可以使用 **R** ETURN 做爲程式結束的檢查字，就如同下面的情形：

```
190  PRINT "ENTER ONE CHARACTER"
200  INPUT C$
205  IF C$ = "" THEN  END
210  FOR I = 1 TO 800
220  PRINT C$;
230  NEXT I
240  PRINT "PHEW!"
250  GOTO 190
```

在行號 205 的敍述中，檢查了 **C $** 是否有一個虛值（**NULL VALUE**），而這種情形只發生在您對行號 200 的 **INPUT** 敍述並不做任何的輸入而只是按下 **R** ETURN 鍵時。

做爲最後的一項工作，您可能重新研究過這個程式並且加入註解，並且對於 **FOR - NEXT** 迴圈內的脚註值 800 做一個說明；您可以在同一行中使用冒號將說明放在 **FOR - NEXT** 迴圈敍述的同一行中：

```
210 FOR I=1 TO 800:REM 800/40=20 LINES
```

加入一行說明，說明一個虛無字的輸入結束程式的執行：

```
203 REM END PROGRAM ON NULL ENTRY
```

最後，您可以在程式的起始處加上幾條程式行說明這個程式的功用，同時給程式一個名稱。最後的程式就如同圖4－3所顯示的。

```
 10 REM ******* BLANKET *******
 20 REM CONTINUOUS-LINE DISPLAY OF ONE
 30 REM CHARACTER ENTERED FROM
 40 REM THE KEYBOARD
 50 REM *********************
190 PRINT "ENTER ONE CHARACTER"
200 INPUT C$
203 REM END PROGRAM ON NULL ENTRY
205 IF C$ = "" THEN END
210 FOR I = 1 TO 800: REM 800/40 = 20 LINES
220 PRINT C$;
230 NEXT I
240 PRINT "PHEW!"
250 GOTO 190
```

圖4－3　BLANKET程式

提示訊息

任何需要由外界輸入資料的程式都應該藉著詢問的方式告訴操作員如何做；問題的型式大都是顯示在同一行中，並且要求一個簡單的是或否的回應；例如，您可以顯示下面的訊息：

```
DO YOU WANT TO MAKE ANY CHANGES?
```

對操作員來講，他必須輸入YES或NO回應這個問題（經常只需要輸入Y或N即可）。另外一個例子就是這個訊息給操作員幾種選擇的機會。下面就是這種訊息：

WHICH ENTRY DO YOU WISH TO CHANGE?

這個訊息可以讓操作員輸入一個代表某項工作的碼。

一個控制上面這種提示訊息的程式應該是一個副程式，而這個副程式對於主程式並不做任何其他的工作，它只是顯示訊息並與主程式做資料的交換工作。下面有三點是必須注意的：

1. 由於您無法確定在提示訊息顯示的位置是否為空白，而且如果它們不是空白，那麼提示訊息將蓋在它們的上面，但是如果提示訊息所遮掩的位置不夠，那麼剩下的字仍然顯示在螢光幕上，這對於操作員來講，很可能造成他們視覺的混淆而導致錯誤發生。下面的副程式依照ER的值把同樣多的位置清為空白。

```
5000  REM  ERASE SPACES
5010  FOR I = 1 TO ER: PRINT " ";: NEXT I: RETURN
```

2. 這個副程式必須由主程式接受訊息；例如，若這個副程式要求操作員輸入一個數字，那麼主程式必須告訴副程式這個數字的上下限各為什麼值。
3. 這個副程式必須把操作員的回應送回主程式，這可能是一個字（例如，Y或N）、一個文字YES或NO或是一個數字。

由於副程式無法推想出提示訊息應該顯示在螢光幕的那一個位置，因此要求主程式在呼叫它之前就已經把游標放在正確位置上，應該是合理的。

現在讓我們看看下面這個程式，其中包含了一個詢問的副程式，這個副程式所需要的回應是Y（YES）或N（NO），我們使用PRINT 敘述問問題，隨後就使用INPUT敘述接受一個字的回應。我們

將使用上面的副程式來清除部份顯示幕的畫面；下面就是這個程式及副程式：

```
100  REM  MOVE CURSOR
140  VTAB 1
150  REM  CALL SUBROUTINE TO GET RESPONSE
160  GOSUB 3020
170  END
3000  REM  ++ GET Y/N RESPONSE ++
3010  REM
3020 C =  POS (0): REM  REMEMBER CURSOR COLUMN
3030 R =  PEEK (37): REM  REMEMBER CURSOR ROW
3040  REM  CLEAR SPACE FOR QUERY
3050 ER = 35: REM  CLEAR 35 SPACES
3060  GOSUB 5010
3070  HTAB C + 1: REM  REPOSITION CURSOR
3080  PRINT "DO YOU WANT TO MAKE ANY CHANGES";
3090  INPUT R$
3100  IF R$ = "Y" OR R$ = "N" THEN  RETURN
3110  REM  IMPROPER RESPONSE
3120  VTAB R + 1: REM  REPOSITION CURSOR (VERT)
3130  GOTO 3050: REM  TRY AGAIN
5000  REM  ++ ERASE SPACES ++
5010  FOR I = 1 TO ER: PRINT " ";: NEXT I: RETURN
```

其次我們考慮使操作員能夠輸入正確數字的句法；我們將設計一個副程式，它可以接受一個不大於 HI 內的値也不小於 LO 內値的數値，而這副程式會把輸入的値放在變數NM中。下面就是整個程式：

```
100  REM  SET RANGE AND POSITION CURSOR
140 LO = 1:HI = 10
150  VTAB 1
160  REM  CALL SUBROUTINE TO GET RESPONSE
170  GOSUB 3500
180  END
3500  REM  ++ GET NUMERIC RESPONSE ++
3510  REM  ++ LO<=RESPONSE<=HI++
3520  REM  ++ NM IS RESPONSE ++
3530  GOSUB 5110: REM  WHERE IS CURSOR NOW?
3540  REM  CLEAR SPACE FOR QUERY
3550 ER = 35: REM  CLEAR 35 SPACES
3560  GOSUB 5010
3570  HTAB C + 1: REM  REPOSITION CURSOR
3580  PRINT "WHICH FIELD DO YOU WANT TO CHANGE (1-10)";
3590  INPUT NM
3600  IF NM > = LO AND NM < = HI THEN  RETURN
3610  REM  IMPROPER RESPONSE
3620  VTAB R + 1: REM  REPOSITION CURSOR (VERT)
3630  GOTO 3550: REM  TRY AGAIN
```

```
5000  REM  ++ ERASE SPACES ++
5010  FOR I = 1 TO ER: PRINT " ";: NEXT I: RETURN
5100  REM  ++ REMEMBER CURRENT CURSOR POSITION ++
5110 C =  PEEK (36): REM  COLUMN
5120 R =  PEEK (37): REM  ROW
5130  RETURN
```

您能改寫上面這個程式使它能接受具有兩位數字的輸入資料嗎？試著去修改這個程式。如果您還不能做這個工作，那麼您將在本章的後半部發現適當的副程式。

這裏還有一個簡單的修正，使前面所提的兩種顯示訊息的方法更爲簡捷；您可以使用一個字串變數將提示訊息放在主程式內，那將使這兩個副程式具有更廣泛的應用能力。那麼您是否能夠重寫這兩個程式使它們能夠接受由主程式傳送來的訊息呢？

錯誤的偵測及控制

如果您想要設計一個完美的程式，那麼您必須對每一個使用者可能發生的錯誤做一番研究，您的程式必須能夠偵測出輸入的錯誤並且強迫使用者重新輸入正確的資料，免得使您的程式不正常地結束。

APPLE Ⅱ 本身具有一些偵錯的能力；例如，在一個數值的 **INPUT A** 敍述內，若輸入文字的資料，**APPLE Ⅱ** 將不接受這個輸入，同時顯示錯誤訊息並要求重新輸入這個值。

機器本身具有的偵錯能力到底是有限的，很可能輸入值的型式是正確的（例如，數字或字串），但是這個值却無法在程式中被接受；也就是說，這個值將使得程式執行到某個地方時就產生了錯誤。下面就是一個說明這種情形的程式：

```
100  INPUT X
200  PRINT 100 / X
300  END
```

如果您輸入ø做爲對INPUT敍述的回應，那麼當程式執行到PRINT敍述中的除法時，程式就無法繼續下去了。這種情形是極爲容易避免的，下面的幾個程式行將檢查輸入的值是否爲ø，若是ø則要求重新輸入。

```
110  IF X < > 0 THEN 200
120  PRINT "NOT ALLOWED...RE-ENTER"
130  GOTO 100
```

將這個例子的原則引伸出來，您可以發現對於檢查是否在規定範圍內的輸入值這個工作是多麼的容易；由於在這種情形下，您可以使用APPLESOFT 中的ON-GOTO或ON-GOSUB敍述（在整數BASIC中就是計算式的 GOTO 或 GOSUB ）代替一連串的 IF-THEN 敍述做範圍檢查的工作。在下一節中我們將顯示一個比較複雜的偵測錯誤程式。

ONERR GOTO及RESUME敍述

APPLESOFT 中有一個特別的敍述允許您在程式發生錯誤時做適當的措施，以免程式停止執行。下面就是一個例子：

```
50  ONERR  GOTO 8000
```

只要像這樣的敍述一被執行，當 APPLESOFT 偵測出錯誤的時候，就會把程式的執行轉移到這個敍述所指出的程式行號去；同樣地，APPLESOFT 也會把一個描述錯誤情形的數值碼放到記憶體位置222的地方去，於是您也就可以利用 PEEK 敍述來觀察這個值。在附錄C的表C－1中詳細列出了在 ONERR GOTO 中偵測出來的錯誤情況。

一般使用 ONERR GOTO 敍述處理錯誤的方法，大都是設計一段處理錯誤的程式，當錯誤發生時，程式的執行就轉移到那段程式去，

在這段程式結束的地方，我們可以使用 RESUME 敍述使得執行轉移回到發生錯誤的敍述的起始處，另外一種方式就是使用 GOTO 敍述將程式的執行轉移到任何您希望的地方去。您可以設計一段處理錯誤的程式使它能夠依照某些關鍵值而對不同的錯誤做不同的反應動作。

您可以使用 POKE 216 、∅敍述回返到APPLE Ⅱ 的自動偵錯情況下，而不使用 ONERR GOTO 的偵錯方式。

下面的程式對使用 ONERR GOTO 敍述做一個示範；在這個程式中如果發生並非由鍵盤輸入所造成的錯誤，將使程式結束並印出適當的忠告訊息，由鍵盤輸入的錯誤則被告知而且要求重新輸入。

```
50  ONERR  GOTO 8000
200  PRINT "ENTER A STRING VALUE"
210  INPUT X$
220  PRINT "ENTER A NUMERIC VALUE"
230  INPUT X
240  PRINT "ENTER AN INTEGER VALUE"
250  INPUT X%
260  GOTO 200
500  REM  END OF PROGRAM
510  PRINT "LAST ENTRIES WERE ";X$;", ";X;" AND ";X%
515  POKE 216,0: REM  TURN OFF ON ERR
520  END
8000  REM  ++ ERROR HANDLING ROUTINE ++
8010 E =  PEEK (222): REM  GET ERROR NO.
8020  IF E = 255 THEN  GOTO 500: REM   END PROGRAM ON CTRL-C
8030  IF E = 53 OR E = 176 OR E = 254 THEN 8100
8035  INVERSE
8040  REM  PROGRAMMING ERROR DETECTED
8050  PRINT "ARRGH! ERROR NO. ";E;" FOUND."
8055  PRINT "WRITE DOWN THIS NUMBER"
8060  PRINT "AND A DESCRIPTION OF WHAT WAS GOING ON."
8070  PRINT "CALL A PROGRAMMER FOR HELP."
8080  PRINT "LEAVE THE COMPUTER ON!"
8090  NORMAL : STOP
8100  REM  INPUT ERROR DETECTED
8110  PRINT "";: REM   CTRL-G CHARACTERS IN QUOTES
8130  PRINT "ERROR...TRY AGAIN"
8140  RESUME
```

輸入一個正確的日期

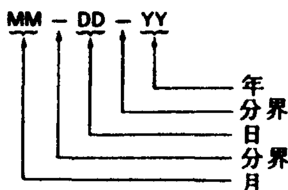
在這一節中我們將使用在本章中所描述的一些敍述發展一個程式，

這個程式使用了一些APPLESOFT中的BASIC函數及敍述。當然您也可以將這個程式改寫爲整數BASIC的程式。

許多的程式在某些地方需要一些簡單的輸入資料——比只是簡單的YES或NO多一些，但比整螢光幕的輸入來得少——例如時間這個輸入。

您必須對這種簡單的輸入多加一些注意，因爲這種時間的輸入可能只是一長串資料輸入中的一小部份，如果您不小心地設計每一個資料的輸入，很可能會使操作員在輸入了許多資料後由於一個小小的錯誤而必須重新來過。

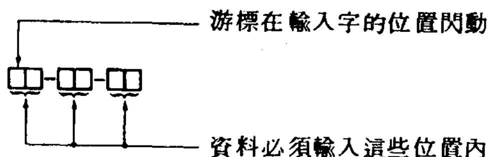
我們假設時間的輸入依照下面的格式：



月份、日期及那一年分別用兩位數字輸入，而不使用 RETURN 做爲結束。

這個程式使用破折號（一）將輸入的資料分開，若是您自己設計的話，那麼可以依您的喜好使用任何符號。而且在世界上的許多地方常常月份放在日期的後面，這也可以依需要而自行設定。

程式中資料輸入的設計必須是使操作員很容易分辨清楚，操作員必須能夠很快地瞭解資料應該由何處輸入，資料的型式是什麼，以及這個輸入資料的過程有多長；將資料輸入的位置用正常顯現的相反方式表示不失爲指示資料應該在何處輸入的一個好方法；例如，這個要求日期資料輸入的程式可以使用上面的方法將下面的格式用黑白相反的方式顯示出來：



而您可以使用下面的程式達到這個目的：

```
10  HOME : VTAB 3: HTAB 20: REM   POSITION FOR INPUT
20 IW = 2: GOSUB 1100: REM  2-CHAR INPUT FIELD
30  PRINT "-";
40  GOSUB 1100: REM  2-CHAR INPUT FIELD
50  PRINT "-";
60  GOSUB 1100: REM  2-CHAR INPUT FIELD
70  VTAB 3: HTAB 20: REM  REPOSITION TO START OF INPUT FIELD
80  END
1090  REM ++ DISPLAY "IW" REVERSE-BLANKS ++
1100  INVERSE
1110  FOR I = 1 TO IW: PRINT " ";: NEXT I
1120  NORMAL
1130  RETURN
```

上面的程式包含了將日期資料的輸入位置放在第 21 行，第三列的敍述，同時並把螢幕上多餘的字消除掉；雖然由於 **END** 敍述的緣故，使得游標移動的情形並不明顯，但是這個程式在把日期輸入的位置顯示出來之後就把游標移到第一個輸入資料的起始位置去了。

試著把**INPUT**敍述加在行號 80 的地方，使得這個程式可以接受這個日期資料的第一部份：月份。這可以由下面的程式行完成：

```
80  INPUT M$
90  END
```

輸入上面的敍述放在行號 80 及 90 的程式行，然後執行這個程式；如您所見到的，**INPUT** 敍述將不會如前所述達成接受月份資料的目的；因此，您可以使用 **GET** 敍述達成接受資料的目的；加入下面的程式行：

```
80  GOSUB 1200:MM$ = C$: REM  GET 1ST MONTH DIGIT
90  GOSUB 1200:MM$ = MM$ + C$: REM  GET 2ND MONTH DIGIT
1190  REM  ++ ACCEPT ONE INPUT CHARACTER ++
1200  GET C$
1210  PRINT C$;: REM  ECHO KEYSTROKE
1220  RETURN
```

上面的這些敍述可以接受兩位數字的輸入，輸入的資料顯示在日期的第一個黑白相反的位置上；而這個兩位數字的輸入並不需要您按 **RETURN** 鍵或是打入任何鍵，程式會在您輸入兩個字之後自動地結束輸入的工作。

現在我們需要三組兩位數字的資料輸入——月份、號數以及年份；我們將這些敍述放在一個副程式中，並且在需要的時候進入這個副程式三次，而不使用 **GOTO** 敍述轉移程式的執行。下面就是這一段程式：

```
80  GOSUB 1300:MM$ = CC$: REM  GET MONTH
90  PRINT "-";: REM  SPACE OVER TO NEXT FIELD
180  GOSUB 1300:DD$ = CC$: REM  GET DAY
190  PRINT "-";: REM  SPACE OVER TO NEXT FIELD
280  GOSUB 1300:YY$ = CC$: REM  GET YEAR
290  END
1290  REM ++ GET A 2-CHARACTER INPUT ++
1300  GOSUB 1200:CC$ = CC$ + C$: REM  GET 2ND CHARACTER
1320  RETURN
```

變數**MM$**、**DD$**及**YY$**分別存入了月份、日數及年份的資料，而每一個輸入都是兩個位置的字串。

下面有兩種方法使我們可以幫助操作員在他們發生輸入錯誤時解決問題。

1. 程式可以自動地測試輸入的資料是否是有效的月份、日數及年份資料。
2. 操作員可以有一種方法重新輸入資料。

這個程式能檢查月份的資料是否在1到12之間；程式將不管閏年，但是它將檢查這個月份的天數；同時，年份的資料必須在00與99之

間，任何不正確的輸入都將使整個日期輸入的程序重新開始。

如果操作員按下 RETURN 鍵，那麼整個輸入日期的程序就重新開始。現在我們最後的輸入日期的程式就如同下面一般：

```
10  HOME : VTAB 3: HTAB 20: REM   POSITION FOR INPUT
15 RC$ =  CHR$ (13): REM   ASSIGN RESTART-ENTRY CHARACTER
17  REM
18  REM  DISPLAY THREE 2-CHAR. INPUT FIELDS
19  REM
20 IW = 2: GOSUB 1100
30  PRINT "-";
40  GOSUB 1100
50  PRINT "-";
60  GOSUB 1100
65  PRINT "";: REM  CTRL-G(BELL)
67  REM
68  REM  GET THREE FIELD ENTRY
69  REM
70  VTAB 3: HTAB 20: REM  REPOSITION TO START OF INPUT FIELD
79  REM
80  REM  GET MONTH
81  REM
90  GOSUB 1300: IF C$ = RC$ THEN  GOTO 10: REM   CHECK FOR
 ENTRY RESTART
100 MM =  VAL (CC$): REM  MONTH NUMBER
110  IF MM < 1 OR MM > 12 THEN 10: REM  CHECK FOR UNREAL
 MONTH
120 DT$ = CC$: PRINT "-";: REM    ADVANCE TO NEXT FIELD
125  REM  DETERMINE MAX. NO. OF DAYS THIS MONTH
130 DM = 31: REM  ASSUME 31 DAYS
135  REM -UNLESS FEBRUARY
140  IF MM = 2 THEN DM = 29
150  REM  OR APR. JUNE. SEPT. NOV
160  IF MM = 4 OR MM = 6 OR MM = 9 OR MM = 11 THEN DM = 30
169  REM
170  REM  GET DAY
171  REM
180  GOSUB 1300: IF C$ = RC$ THEN  GOTO 10: REM   CHECK FOR
 ENTRY RESTART
190 DD =  VAL (CC$): IF DD < 1 OR DD > DM THEN 10: REM
 RESTART IF ENTRY IS INVALID
200 DT$ = DT$ + "-" + CC$: PRINT "-";: REM  ADVANCE TO
 NEXT FIELD
269  REM
270  REM  GET YEAR
271  REM
280  GOSUB 1300: IF C$ = RC$ THEN  GOTO 10: REM   CHECK FOR
 ENTRY RESTART
290 YY =  VAL (CC$): IF YY < 0 OR YY > 99 THEN 10: REM
 RESTART IF ENTRY IS INVALID
300 DT$ = DT$ + "-" + CC$
339  REM
```

```
390  REM  DISPLAY ENTRY
391  REM
400  VTAB (10): HTAB (18): PRINT "DATE ENTERED:"
410  VTAB (11): HTAB (20): PRINT DT$
420  END
1089  REM
1090  REM  ++  DISPLAY 'IW' REVERSE-BLANKS ++
1091  REM
1100  INVERSE
1110  FOR I = 1 TO IW: PRINT " ";: NEXT I
1120  NORMAL
1130  RETURN
1189  REM
1190  REM  ++ ACCEPT ONE CHARACTER INPUT ++
1191  REM
1200  GET C$: IF C$ = "" THEN 1200
1210  IF C$ = RC$ THEN  RETURN : REM  CHECK FOR RESTART
1220  IF C$ < "0" OR C$ > "9" THEN  GOTO 1200
1230  PRINT C$;: REM   ECHO KEYSTROKE
1240  RETURN
1289  REM
1290  REM  ++ GET A 2-CHAR. INPUT ++
1291  REM
1300  REM  GET 1ST CHARACTER: CHECK FOR RESTART
1310  GOSUB 1200: IF C$ = RC$ THEN  RETURN
1315 CC$ = C$
1320  REM   GET 2ND CHARACTER: CHECK FOR RESTART
1330  GOSUB 1200: IF C$ = RC$ THEN  RETURN
1335 CC$ = CC$ + C$
1340  RETURN
```

要注意的是，日期是放在具有八個字長度的 **DT$** 字串中，包含了月份、號數及年份。

當資料輸入時程式中自動地做下面三種檢查工作：

1. 輸入的字是否是一個游標回頭（ **CARRIAGE RETURN** ）？
2. 如果不是游標回頭，那麼是否是一個有效的數字？
3. 是否這種兩個字的結合是一個有效的月份？而第二組的輸入是否爲一個有效的號數？第三個輸入是否爲一個有效的年份？

我們選擇游標回頭做爲程式重新開始的標示，您可以將行號 15 的

CHR$(13)用任何其他的字代替，取代游標回頭。當操作員按下這個被選擇的鍵，那麼整個程式就重新開始了；由於我們希望在輸入一個字或是輸入兩個字時都可以重新開始，因此我們必須使用一個檢查一個字的副程式（在行號 1200的地方），同時再在輸入兩個字的時候再做一次檢查的工作（在行號1300 的地方），主程式也檢查重新開始的字，使得在碰到這個字的時候，程式的執行可以轉移到行號 10 的地方去，而重新開始輸入資料。當然，您也可以由輸入一個字的副程式直接轉移到行號 10 的地方去，那使得其他的測試都不被執行了，但是這麼做實在是一個不好的習慣，因為使用 GOTO 敍述而不使用RETURN敍述將程式的執行由副程式轉移出去，是一個很危險的事；每一個副程式都應該被視為一個邏輯的完整體系，都具有指定的進入點及標準的副程式回返，任何其他方式的邏輯轉移只會造成程式的複雜性及難以避免的錯誤（記得如果您不使用RETURN敍述將程式的執行轉移出副程式，那麼您必須使用POP 敍述將正常RETURN情況下回返的位置消除）。

程式邏輯在一個數字的輸入副程式中做輸入資料是否為數字的測試，我們選擇不接受非數字資料的輸入，而這項工作由行號1220 的敍述執行。

檢查有效的月份、號數及年份的程式行必須在主程式之內（不在副程式中），因為它們分別有它們各自的極限值。

行號 110 的敍述測試有效的月份資料。

行號 130 、140 及 160 的敍述計算前面輸入月份所能容許的日數，行號 190 的敍述則檢查是否為一個有效的號數。

檢查輸入的年份是否有效是非常簡單的，行號 290 的敍述就是做這個工作。

想要設計一個好的資料輸入的程式，使得顯示的訊息使人易於接受而又能檢查錯誤的輸入，並使操作員能隨時更改輸入的資料，實在需要許多額外的時間；那麼所花的這些時間是否值得呢？答案當然是肯定的，因為您可能只需要設計這個程式一次，但是操作員可能要使用它上百

次或者上千次，因此，您只多花了一次額外的時間，而使得操作員免掉了幾百次或者幾千次的延誤。

編造資料輸入的格式

在這一節中我們將描述一些設計程式的技巧，這些技巧大都是只能在 APPLESOFT 中應用，因爲有許多的效果在整數 BASIC 中是很難或者根本是不可能得到的；如果您只能使用整數 BASIC，那麼您也必須閱讀這一節，因爲有一些的技巧也可以改變成整數 BASIC 的。

要處理許多項目的資料輸入最好的方法就是先造成一個格式（FORM ），然後資料完全依照格式而輸入。試想一個名字及地址，首先顯示出下面的格式：

```
ENTER NAME AND ADDRESS BELOW
1 NAME:
2 STREET:
3 CITY:
4 STATE:
5 ZIP:
```

您應該注意到每個輸入都被給了一個數目，而在這個格式中，數字是用黑底白字的方式顯現的。

操作員依照順序輸入資料——由項目 1 到項目 5，而且可以在輸入完畢之後更改任何項目的資料輸入。

下面的程式將會把螢幕清除並把上面的格式顯示出來：

```
109  REM
110  REM DISPLAY THE DATA ENTRY FORM
111  REM
120  CALL  - 936: VTAB 2: REM  CLEAR SCREEN AND POSITION
 CURSOR
125  PRINT "ENTER NAME AND ADDRESS BELOW"
130  REM FIRST DISPLAY FIELD NUMBERS
140  INVERSE
150  FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160  VTAB 6: HTAB 29: PRINT 5
```

```
170   NORMAL
180   REM NOW DISPLAY FIELD NAMES
190   VTAB 3: HTAB 6: PRINT "NAME:"
200   HTAB 4: PRINT "STREET:"
210   HTAB 6: PRINT "CITY:"
220   HTAB 5: PRINT "STATE:";
230   HTAB 31: PRINT "ZIP:"
```

當每一個資料都被輸入後，我們使用黑白相反的顯示方式指明資料輸入的地方。C_{TRL}-X 被用來重新輸入目前輸入的項目，而R_{ETURN} 鍵被使用代表這個項目輸入工作的完成。下面的程式順序提供了必須的程式邏輯：

```
100 RC$ =  CHR$ (24): REM  CTRL-X IS THE RESTART CHAR.
299  REM
300  REM ENTER ALL 5 FIELDS
301  REM
310  FOR F = 1 TO 5: GOSUB 1900: NEXT F
320  END
990  REM ++++++++++SUBROUTINE 1000++++++++++
991  REM  ENTER STRING DATA INTO A FIELD WITH LN CHARACTERS
992  REM THE CURSOR MUST BE IN THE FIELD'S FIRST POSITION
993  REM THE RETURN KEY WILL END DATA ENTRY
994  REM THE 'LEFT ARROW' KEY RESTARTS ENTRY
995  REM NO VALIDITY CHECKS ON ENTERED DATA
996  REM THE ENTERED STRING IS RETURNED IN CC$
997  REM
1000 HT =  POS (0) + 1: REM   REMEMBER START-OF-FIELD
 POSITION
1010  REM  DISPLAY INVERSE VIDEO ENTRY MASK
1020  INVERSE
1030  FOR I = 1 TO LN: PRINT " ";: NEXT I
1040  NORMAL : HTAB (HT): REM  REPOSITION TO START OF FIELD
1050  REM  ENTER DATA
1060 CC$ = "": REM  INITIALIZE OUTPUT TO NULL
1070  GET C$
1080  IF C$ = RC$ THEN  HTAB (HT): GOTO 1020: REM  RESTART
 ENTRY?
1090  IF C$ =  CHR$ (13) THEN  GOTO 1140: REM  END OF ENTRY?
1100  REM   WHEN ENTRY IS FULL, WAIT FOR RETURN OR RESTART
1110  IF  LEN (CC$) = LN THEN  GOTO 1070
1120  PRINT C$;: REM  ECHO KEYSTROKE
1130 CC$ = CC$ + C$: GOTO 1070
1135  REM   ENTRY FINISHED, FILL THE REST OF CC$ WITH BLANKS
1140 J =  LEN (CC$)
1150  FOR I = J TO LN:CC$ = CC$ + " ": NEXT I
1160  REM REDISPLAY ENTRY
1170  HTAB (HT): PRINT CC$;: RETURN
1889  REM ++++++++++SUBROUTINE 1900++++++++++
1890  REM   BRANCH TO ENTRY ROUTINE FOR FIELD NUMBER F
```

```
1891  REM
1900  ON F GOTO 2000,2100,2200,2300,2400: RETURN
1989  REM
1990  REM ENTER 20-CHAR. NAME
1991  REM
2000  VTAB 3: HTAB 11
2010 LN = 20: GOSUB 1000:NA$ = CC$: RETURN
2089  REM
2090  REM  ENTER 20-CHAR STREET
2091  REM
2100  VTAB 4: HTAB 11
2110 LN = 20: GOSUB 1000:S1$ = CC$: RETURN
2189  REM
2190  REM  ENTER 20-CHAR. CITY
2191  REM
2200  VTAB 5: HTAB 11
2210 LN = 20: GOSUB 1000: RETURN
2289  REM
2290  REM  ENTER 18-CHAR. STATE
2291  REM
2300  VTAB 6: HTAB 11
2310 LN = 18: GOSUB 1000:ST$ = CC$: RETURN
2389  REM
2390  REM  ENTER 5-CHAR. ZIP CODE.
2391  REM
```

輸入由行號100到結束的所有敍述，並且執行它，如果您在前面那個例子中已經輸入了由行號109到230的敍述，那麼您在輸入這個程式時，就不必重新打入了。

如果您的程式執行並不正確，那麼請您仔細地檢查一下，尤其注意到**PRINT**敍述中的分號。

當您執行這個程式時，這五個區域將會輪流地發閃光，而當您輸入資料時，這些字將被顯示在這五個區域內，當您打入RETURN鍵後，這緊個黑白相反的區域就被您輸入的資料所代替了。您也可以試著使用**CTRL-X**重新開始輸入資料。

仔細地研究一下由行號1000到1170的輸入字串的副程式，在你更進一步研究之前，您必須對這個程式的邏輯非常瞭解。

您注意到想要看到您所輸入的資料是一件多麼容易的事了吧！而且重新改正輸入資料的錯誤又是多麼的簡單。

在完整的名字及地址被輸入之後，程式應該詢問操作員是否需要修

改任何區域，然後再詢問那一個區域是要被修改的。我們在前面曾經介紹過一個詢問是或否（Y或N）的副程式，我們在這裏將它稍稍的修改一下，由主程式提供詢問操作員的問題。下面就是這整個的程式：

```
9  REM *******************************************************
10  REM THIS PROGRAM DISPLAYS A FORM FOR ENTERING A NAME AND
11  REM ADDRESS THEN IT REQUESTS ENTRY OF THAT DATA
12  REM *******************************************************
13  REM
100 RC$ =  CHR$ (24): REM  CTRL-X IS THE RESTART CHAR.
109  REM
110  REM DISPLAY THE DATA ENTRY FORM
111  REM
120  CALL  - 936: VTAB 1: REM  CLEAR SCREEN AND POSITION
 CURSOR
125  PRINT "ENTER NAME AND ADDRESS BELOW"
130  REM FIRST DISPLAY FIELD NUMBERS
140  INVERSE
150  FOR I = 1 TO 4: HTAB 2: PRINT I: NEXT I
160  VTAB 6: HTAB 29: PRINT 5
170  NORMAL
180  REM NOW DISPLAY FIELD NAMES
190  VTAB 3: HTAB 6: PRINT "NAME:"
200  HTAB 4: PRINT "STREET:"
210  HTAB 6: PRINT "CITY:"
220  HTAB 5: PRINT "STATE:";
230  HTAB 31: PRINT "ZIP:"
299  REM
300  REM ENTER ALL 5 FIELDS
301  REM
310  FOR F = 1 TO 5: GOSUB 1900: NEXT F
319  REM
320  REM  ALLOW CHANGES
321  REM
330  VTAB 23: HTAB 1: REM  GET ENTRY ON BOTTOM LINE
340 QU$ = "DO YOU WANT TO MAKE ANY CHANGES? "
350  GOSUB 1300: REM GET Y/N RESPONSE
360  IF YN$ = "N" THEN  GOTO 500
370  VTAB 23: HTAB 1: REM GET ENTRY ON BOTTOM LINE
380 QU$ = "ENTER NUMBER OF FIELD TO CHANGE "
390 LO = 1:HI = 5
400  GOSUB 1400: REM GET NUMERIC RESPONSE
410 F = NM: GOSUB 1900: REM CHANGE FIELD F
420  GOTO 330
490  REM  END OF PROGRAM
500  VTAB 23: HTAB 1: GOSUB 1200: REM CLEAR BOTTOM LINE
510  END
990  REM +++++++++++SUBROUTINE 1000+++++++++++
991  REM  ENTER STRING DATA INTO A FIELD WITH LN CHARACTERS
992  REM THE CURSOR MUST BE IN THE FIELD'S FIRST POSITION
993  REM THE RETURN KEY WILL END DATA ENTRY
994  REM THE 'LEFT ARROW' KEY RESTARTS ENTRY
995  REM NO VALIDITY CHECKS ON ENTERED DATA
```

```
996  REM THE ENTERED STRING IS RETURNED IN CC$
997  REM
1000 HT =  POS (0) + 1: REM  REMEMBER START-OF-FIELD POSITION
1010  REM  DISPLAY INVERSE VIDEO ENTRY MASK
1020  INVERSE
1030  FOR I = 1 TO LN: PRINT " ";: NEXT I
1040  NORMAL : HTAB (HT): REM  REPOSITION TO START OF FIELD
1050  REM  ENTER DATA
1060 CC$ = "": REM  INITIALIZE OUTPUT TO NULL
1070  GET C$
1080  IF C$ = RC$ THEN  HTAB (HT): GOTO 1020: REM  RESTART
 ENTRY?
1090  IF C$ =  CHR$ (13) THEN  GOTO 1140: REM  END OF ENTRY?
1100  REM   WHEN ENTRY IS FULL, WAIT FOR RETURN OR RESTART
1110  IF  LEN (CC$) = LN THEN  GOTO 1070
1120  PRINT C$;: REM  ECHO KEYSTROKE
1130 CC$ = CC$ + C$: GOTO 1070
1135  REM   ENTRY FINISHED, FILL THE REST OF CC$ WITH BLANKS
1140 J =  LEN (CC$)
1150  FOR I = J TO LN:CC$ = CC$ + " ": NEXT I
1160  REM REDISPLAY ENTRY
1170  HTAB (HT): PRINT CC$;: RETURN
1189  REM +++++++++++SUBROUTINE 1200+++++++++++
1190  REM  CLEAR ROW WHICH THE CURSOR IS ON
1191  REM
1200  HTAB 1: REM START AT BEGINNING OF ROW
1210  FOR I = 1 TO 39: PRINT " ";: NEXT I
1220  HTAB 1: REM LEAVE CURSOR AT BEGINNING OF ROW
1230  RETURN
1289  REM +++++++++++SUBROUTINE 1300+++++++++++
1290  REM ASK A QUESTION (QU$) AND RETURN A Y OR N RESPONSE
 IN YN$
1291  REM
1300  GOSUB 1200: REM  CLEAR ENTRY LINE
1310  PRINT QU$;: REM  DISPLAY PROMPT
1320  GET YN$: IF YN$ < > "N" AND YN$ < > "Y" THEN  GOTO
 1320
1330  PRINT YN$;: REM ECHO RESPONSE
1340  RETURN
1389  REM +++++++++++SUBROUTINE 1400+++++++++++
1390  REM ASK FOR NUMERIC ENTRY (PROMPT IS QU$)
1391  REM RETURN RESPONSE IN NM
1392  REM NM MUST BE <=HI AND >=LO
1393  REM
1400  GOSUB 1200: REM  CLEAR ENTRY LINE
1410  PRINT QU$;: REM DISPLAY PROMPT
1420  GET C$:NM =  VAL (C$)
1425  REM  CHECK THAT ENTRY IS WITHIN RANGE
1430  IF NM < LO OR NM > HI THEN  GOTO 1420
1440  PRINT C$;: REM  ECHO RESPONSE
1450  RETURN
1889  REM +++++++++++SUBROUTINE 1900+++++++++++
1890  REM   BRANCH TO ENTRY ROUTINE FOR FIELD NUMBER F
1891  REM
```

```
1900  ON F GOTO 2000,2100,2200,2300,2400: RETURN
1989  REM
1990  REM ENTER 20-CHAR. NAME
1991  REM
2000  VTAB 3: HTAB 11
2010 LN = 20: GOSUB 1000:NA$ = CC$: RETURN
2089  REM
2090  REM  ENTER 20-CHAR STREET
2091  REM
2100  VTAB 4: HTAB 11
2110 LN = 20: GOSUB 1000:CI$ = CC$: RETURN
2189  REM
2190  REM  ENTER 20-CHAR. CITY
2191  REM
2200  VTAB 5: HTAB 11
2210 LN = 20: GOSUB 1000: RETURN
2289  REM
2290  REM   ENTER 17-CHAR. STATE
2291  REM
2300  VTAB 6: HTAB 11
2310 LN = 17: GOSUB 1000:ST$ = CC$: RETURN
2389  REM
2390  REM  ENTER 5-CHAR. ZIP CODE
2391  REM
2400  VTAB 6: HTAB 35
2410 LN = 5: GOSUB 1000:ZI$ = CC$: RETURN
```

您應該仔細地研究這個關於名字與地址的程式，而且對於資料輸入的目的也要能夠瞭解。它們分別是：

1. 利用標示每一個區域及在適當的時機使用黑白相反的顯示方法，您可以明白地告訴操作員，什麼樣的資料是這個程式所期待的，以及有多少位的輸入位置可供輸入。
2. 當操作員輸入了代表那一個區域的號碼，以便作更改時，黑白相反的區域能夠很快地告訴操作員是否輸入了正確的選擇號碼。
3. 操作員不必將所有的位置全部填滿，當他或她按下了RETURN鍵之後，剩下的空位全部填入空白。
4. 在輸入一組資料的任何時候，操作員都可以使用CTRL-X重新輸入資料。
5. 當程式在詢問操作員問題時，只有一些具有意義的反應是被

接受的：如Y或N分別代表YES及NO，或者用1到5的數字分別代表五個不同的選擇區域。例如，若使用Y代表YES而使用其他任何的鍵代表NO，那麼很可能引起很大的混亂，因爲當操作員不小心打錯了鍵時，很可能使得資料的輸入中途中斷。反過來說，如果使用N代表NO，而使用其他任何的鍵代表YES，那麼很可能使得操作員做不必要的重複輸入，原因只是因爲不小心按錯了鍵。

下面是一些我們沒有加入但是可以增加的資料輸入的特性：

1. 對於非數字的輸入可以檢查ZIP碼（有一些地區允許非數字的ZIP碼，ZIP就是郵遞區號）。
2. 許多謹慎的程式設計師會問這樣的問題：ARE YOU SURE?；當操作員對於DO YOU WANT TO MAKE ANY CHANGES?的反應是NO的時候，這給操作員一個改變主意的機會。
3. 提供一個額外的方法使得操作員能夠放棄現在輸入的資料而將原先的資料存回去。例如，在我們上面這個範例程式中，如果操作員選錯了要改的區域，那麼他或她必須重新輸入這個區域的資料。這個程式應該可以簡單地辨認一個鍵，而將剛輸入的資料放棄，重新採用原先的資料。
4. 能夠使用←鍵做爲復位鍵，每當←鍵被按下一次時，游標就往左移動一位，而最後被輸入的字在顯示幕上就被黑白相反的底色所代替，同時在輸出字串CC$中就被一個空白所代替。當然，當游標在輸入區域的最起始處時，復位鍵並不發生作用。

您可以試著將上面的一些特性加入前面的範例程式中。

定輸出的格式

當您打開APPLE Ⅱ的開關後，輸出就自動地顯示在螢光幕上；有一些敍述可以將輸出送到列表機或是任何可以接受輸出的設備上。

當您拿到螢光幕的輸出與到列表機上的輸出相比較時，您將發現有許多的不同；例如，列表機也許比螢光幕來得寬，那麼輸出到列表機上的一行，也就超過了螢光幕上一行的位置了；另一方面說來，您使用HTAB 及VTAB（在整數BASIC中就是TAB）可以將游標在螢光幕上自由地移動，但它們就無法將列表機的列表頭在紙上自由地移動。

也有許多的程式技巧在做螢光幕及列表機輸出時，大致是相同的，下面大部份的討論都是相關於兩者都具有的特性，任何只能做螢光幕輸出的技巧都會有特別的說明。如果您計劃設計一個將資料輸出到列表機上的程式，那麼您應該研讀在本章後面討論到如何使用列表機的部份。

設計輸出遠較設計資料輸入來得簡單，因爲並不需要考慮到與操作員溝通的問題，您只需要考慮到輸出的訊息是否是容易被使用的也就夠了。下面就是一些您必須遵守的規則：

1. 避免將許多的訊息擠在很小的空間內。
2. 如果數字或字串是依行的方式排列的，那麼您必須將資料排列整齊，使得看的時候能夠很快地掃瞄過去。
3. 使用黑白相反的方式，將關鍵資料、標題及旁註明確地顯示出來（只能使用在螢光幕輸出時）。

下面是一些指引，對避免一些設計輸出時的錯誤會有很大的幫助。

1. 記得在PRINT敍述中使用分號將需要連接在一起的單獨項目連在一起，這在許多輸出的設計上是一個常犯的錯誤。

2. 在做任何設計之前，先設計您的螢幕或報表格式，您可以使用一些格式紙先做這個工作；在附錄L中有一個顯示螢光幕的格式，使得您可以精確地計算行及列。至於另外一種方法就是試驗然後改正，這種方法往往會比用畫圖的方式花更多的時間。

3. 特別小心不能被行數整除的矩陣腳註。例如，假設您有 25 個項目放在矩陣N$（I）中，而您想將它們印在三行中，那麼您也許會設計下面這樣的程式：

```
100 FOR I = 1 TO 25 STEP 3
200 REM PROCESS COLUMN 1
  .
  .
  .
300 REM PROCESS COLUMN 2
  .
  .
  .
400 REM PROCESS COLUMN 3
  .
  .
  .
500 NEXT I
```

但是在FOR- NEXT 迴圈的最後一個循環時，腳註 26 及 27 雖然並不存在仍然也會被算到。而您可以使用下面的方法檢查 FOR- NEXT 迴圈的結束：

```
100 FOR I = LO TO HI STEP ST
  .
  .
  .
350 I = I + 1
360 IF I > HI THEN 510
  .
  .
  .
500 NEXT
510 REM CONTINUE WITH PROGRAM
```

將資料顯示在螢光幕上

當處理大筆的資料時，一個常用的技巧就是把螢光幕當做是一個資料幕（DATA WINDOW），在任何時刻螢光幕只是顯示需要的資料。從事這種工作的一個方法就是將資料編成頁（PAGES）的方式，一頁剛好是一個螢光幕的大小，使用這種技巧的程式必須有分離的一段程式，它專門顯示每一頁資料的標題及資料，也必須具有使操作者能自由選擇那一頁資料顯示的能力。

我們經常使用矩陣儲存大筆的資料；在這種情形時，您使用螢光幕如同使用照像機的取景器一般，想像矩陣中的資料被寫在一個很大的黑板上，而您使用照像機的取景器觀看這些資料；雖然由於這個黑板大得使您無法在取景器中完全看到，但是您可以藉著上、下、左、右的移動取景器觀看黑板上的任何一部份，而這個螢光幕的功能就如同取景器一般。

我們將使用 APPLESOFT 語言設計一個範例將螢光幕當做一個資料幕，而使用的觀念就是前面描述的取景器的概念。在開始時，我們將建立一個二階的整數矩陣，在每一個矩陣元素中，我們將放入一個代表矩陣腳註的四位數數字，就如下面的情形一般：

X%(*i*,*j*)=0*i*0*j*

例如：

```
X%(3,2)=0302
X%(19,8)=1908
X%(11,12)=1112
```

我們可以如下地設定這個整數矩陣的初值：

```
10   DIM X%(14,50)
20   FOR I = 1 TO 14
30   FOR J = 1 TO 50
```

```
40 X%(I,J) = I * 100 + J
50  NEXT J
60  NEXT I
```

現在我們將顯示出這個矩陣的一部份，我們使用頭兩列做爲標題，使用每一列的前面九位做爲旁註。見以下的情形：

```
             COLUMN    COLUMN    COLUMN
                 XX        XX        XX
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
   ROW YY
```

確實的行數就顯示在 XX 的下面，而確實的列數就顯示在 YY 的那一列。下面就是將行及列的標題用黑白相反方式顯示出來的程式敍述：

```
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + I * 10: PRINT "COLUMN";
1040  NEXT I
1050  PRINT
1060  FOR I = 0 TO 2
1065  REM  1 EXTRA SPACE AHEAD OF 1-DIGIT NOS.
1070 S% = 0: IF C% + I < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT  SPC( S%);C% + I;
1090  NEXT I
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM  1 EXTRA SPACE AHEAD OF 1-DIGIT NOS.
1120 S% = 0: IF I < 10 THEN S% = 1
```

```
1130  HTAB 4: PRINT "ROW";: HTAB 8: PRINT  SPC( S%);I
1140  NEXT I
1150  NORMAL : RETURN
```

我們故意地建立一個較全螢光幕爲小的幕，這樣使我們能對闡述資料幕的概念更爲清楚。若您想要將整個螢幕都用做資料幕，那當然是可以的，但是有時候也許您需要較小的資料幕，使得其他的資料也能夠同時顯現。

我們現在加入要求操作員輸入最小矩陣行號及列號的指令，具有這個行號及列號的矩陣元素將顯示在螢光幕的左上角，而螢光幕將被連串的矩陣元素所塡滿。下面是這個完整的程式：

```
5  REM WINDOW AN A TABLE DISPLAY PROGRAM
6  REM ***************************************************
10  HOME : PRINT "PLEASE WAIT...INITIALIZATION IN PROCESS";
20  DIM X%(14,50)
30  FOR I = 1 TO 14
40  FOR J = 1 TO 50
50 X%(I,J) = I * 100 + J
60  NEXT J
70  NEXT I
75  HOME
80  HTAB 1: VTAB 20: INPUT "ENTER COLUMN (1 TO 12):";C%
90  IF C% < 1 OR C% > 12 THEN  GOTO 80
100  VTAB 21: INPUT "ENTER ROW (1 TO 41):";R%
110  IF R% < 1 OR R% > 41 THEN  GOTO 100
120  VTAB 1: HTAB 1: GOSUB 1000: REM DISPLAY HEADINGS
130  REM FILL IN WINDOW VALUES
135  VTAB 3
140  FOR I = R% TO R% + 9
150  HTAB 10
160  FOR J = C% TO C% + 2
165  REM DISPLAY RIGHT-JUSTIFIED VALUES IN WINDOW
170 X$ =  STR$ (X%(J,I)): PRINT  SPC( 10 -  LEN (X$));X$;
180  NEXT J
190  PRINT : REM NEXT DISPLAY LINE
200  NEXT I
210  VTAB 22: PRINT "CONTINUE? ENTER Y OR N ";
220  GET C$: IF C$ <  > "Y" AND C$ <  > "N" THEN 220
230  IF C$ = "Y" THEN  GOTO 80
240  END
990  REM
991  REM ++++++++SUBROUTINE 1000+++++++++++++
992  REM DISPLAY ROW AND COLUMN HEADINGS
1000  INVERSE
1020  FOR I = 1 TO 3
1030  HTAB 4 + I * 10: PRINT "COLUMN";
1040  NEXT I
1050  PRINT
```

```
1060  FOR I = 0 TO 2
1065  REM 1 EXTRA SPACE AHEAD OF 1-DIGIT NOS.
1070 S% = 0: IF C% + I < 10 THEN S% = 1
1080  HTAB 18 + I * 10: PRINT  SPC( S%);C% + I;
1090  NEXT I
1100  PRINT
1110  FOR I = R% TO R% + 9
1115  REM 1 EXTRA SPACE AHEAD OF 1-DIGIT NOS.
1120 S% = 0: IF I < 10 THEN S% = 1
1130  HTAB 4: PRINT "ROW";: HTAB 8: PRINT  SPC( S%);I
1140  NEXT I
1150  NORMAL : RETURN
```

輸入這個程式並執行它，如果您輸入時沒有發生錯誤，那麼您所注意到的第一件事就是電腦似乎停下來了一會，那是因爲執行行號 30 到行號 40 的 **FOR - NEXT** 迴圈的緣故，這個過程將佔用五到十秒的時間將數字填入 **X%** 矩陣內；這時電腦將在螢幕上顯示一段訊息說明這個工作。如果沒有這段訊息，很可能使用這個程式的人認爲電腦不發生動作了呢！因此當相似的情形發生的時候，顯示一段說明的文字不失爲一個好方法。

需要注意的是，行數可以由 1 到 12 ，而顯示幕具有三行的位置，因此一直到 12 的行數都可以包含在矩陣行數（14）之內；列數可以由 1 到 41 ，這是因爲由 41 開始到 50的 10 行它是第一個矩陣的數目。

矩陣 **X %**中的整數值在行號 170 的地方被轉換成字串，以便簡化顯示的格式；做了這樣的轉換後，對於計算行與行間的空白數就非常容易了，而這個過程就在行號 170 的 **PRINT**敍述中。當您想要將數字排成一條直線般地直接印出來，並不是一件容易的事。將行號 170 改爲下面的敍述，您可以很快的瞭解：

```
170 PRINT SPC <7>; X% <J, I>;
```

如果您想要顯示的數字不超過四位，那麼數字就能夠排成一條直線地顯示出來，但是如果超出了四位數，那麼這個顯示就超過了顯示幕的極限了。

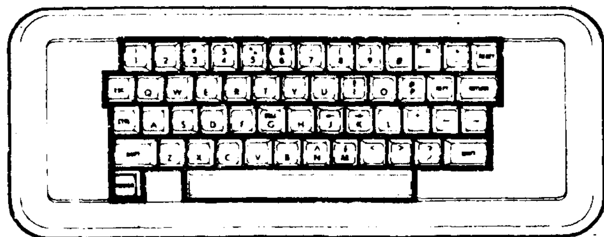
我們的程式在顯示的時候儘量不使它超過螢光幕的第 39 行的位置

，如果您將顯示的資料超過了 40 個字，那麼超過的字就會自動地被移到下一顯示行的位置。您最好不要將顯示的資料超過一個顯示行的長度，因爲這樣往往會使程式中設計的游標回頭與機器自動設定的游標回頭（一行超過 40 字）相混合，造成許多不必要的麻煩。

您可以修改這個程式使它能夠使用到第 40 個字的位置，以便做爲一個練習。您必須將行號150的**HTAB** 10 敍述改爲**HTAB** 11 ，在行號1030的敍述中將 4＋1＊10 改爲 5＋1＊10 ，行號 1080的敍述改爲 19＋1＊10，並將行號 1130中的 4 及 8 改爲 5 及 9 。現在您可以試著執行這個程式，您將發現數字顯示行的位置往上移了一位，但是執行的結果有太多的游標回頭，而且在最上面的行數被第一列的矩陣值掩蓋住了；您可以試試看能否將多餘的游標回頭消除，並且能夠正確地顯示出資料。但是這並不是一個簡單的程式設計工作。

在行號 80 、 100 及 220 的敍述中都要求輸入資料，但需要注意的是，這些敍述之後都跟隨著檢查的工作，不容許無意義的輸入。

對於一個顯示字幕的格式而言，最好的修正就是能使顯示矩陣的字幕能夠上下左右地移動，我們使用**I** 、**J** 、**K** 、**M**鍵直接地控制這個功能（就如同在編修情況下的**I** 、**J** 、**K** 、**M**鍵一般），**I** 鍵使字幕往上移動一列，**M**鍵往下移動一列，**J**鍵則使字幕往左移動一行，**K**鍵則往右移動一行；就如下面顯示的情形一般：



要想達成上面的這個工作，我們必須使用下面的敍述代替由行號

210 到 240 的敘述：

```
210  VTAB 22: PRINT "CONTINUE?"
215  PRINT "ENTER DIRECTION (I,J,K,M), Y, OR N ";
220  GET C$
225  REM DOWN ONE ROW?
230  IF R% > 1 THEN  IF C$ = "M" THEN R% = R% - 1: GOTO 120
235  REM UP ONE ROW?
240  IF R% < 41 THEN  IF C$ = "I" THEN R% = R% + 1: GOTO 120
245  REM LEFT ONE COLUMN?
250  IF C% > 1 THEN  IF C$ = "J" THEN C% = C% - 1: GOTO 120
255  REM RIGHT ONE COLUMN?
260  IF C% < 12 THEN  IF C$ = "K" THEN C% = C% + 1: GOTO 120
270  IF C$ = "Y" THEN  GOTO 80: REM ENTER NEW ROW AND COLUMN
280  IF C$ = "N" THEN  END
285  REM   SOUND BELL AND REJECT ANY OTHER ENTRY
290  PRINT  CHR$ (7);: GOTO 220
```

您應該注意到了這個程式的邏輯是多麼直接了吧！任何不是被允許的六種輸入的字都將不被接受，而且如果控制想要將字幕移到矩陣的範圍之外，也將不被接受。

利用程式控制列表機

APPLE Ⅱ 將列表機視爲顯示螢光幕的代用品，因此，若您想將輸出轉移到列表機上，您必須使用一些敘述從事這個工作，而當輸出到列表機的工作完成後，必須使用 **PR#** 的敘述將控制轉移回螢光幕上。

列表機由於種類的不同，分別可以用序列輸出介面卡或平行輸出介面卡與 **APPLE Ⅱ** 連接。

正常的情況下，序列輸出介面卡大都插在 **APPLE Ⅱ** 的第一個擴充接點上，而平行輸出介面卡則插在第二個擴充接點，但是這種情形完全依照列表機的規格而定，並沒有一定的規則；事實上，不論序列輸出介面卡或平行輸出介面卡都可以插在 **APPLE Ⅱ** 的任何擴充接點上（除了第 0 個擴充接點外）。

將文字資料輸出到列表機上

您也許記得當DOS 在記憶體中時，PR# 似乎是一個 DOS 的敍述，這也就是說，它必須與C_{TRL}-D 聯合使用。下面的程式將把兩行的文字資料印出來，列表機與位於擴充接點 1 上的介面卡相連，而且DOS 目前在APPLE Ⅱ 中：

```
 10 REM  OUTPUT TWO LINES OF TEXT TO A PRINTER
 20 REM  CREATE A CTRL-D CHARACTER
 30 D$=""
 40 REM  SELECT THE SERIAL I/O PORT
 50 PRINT D$;"PR#1"
 60 PRINT "TO A SCREEN OR THE PRINTER AN APPLE WILL WRITE"
 70 PRINT "AND IN EACH CASE THE DATA GOES OUT BYTE BY BYTE"
 80 REM  DESELECT THE PRINTER
 90 PRINT D$;"PR#0"
100 END
```

在行號 30 的敍述中，建立了一個控制的字，而這個控制字（CONTROL CHARACTER）將 PR# 這個命令轉換成BASIC的敍述；我們在這個程式中見到行號 50 及 90 的地方都有 PR# 的敍述，行號 50 的敍述將輸出由螢光幕轉移到列表機上，而在行號 90 的敍述則將輸出由列表機轉回到螢光幕去；在行號 60 及 70 的PRINT 敍述則將兩行文字資料列印到列表機上。下面就是這個程式被執行後的輸出情形：

```
TO THE SCREEN OR A PRINTER THE APPLE WILL WRITE
AND IN EACH CASE THE DATA GOES OUT BYTE BY BYTE
```

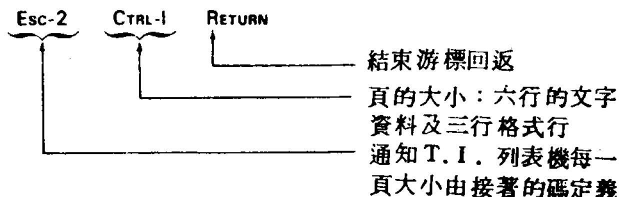
下面是當 DOS 不在記憶體中的同樣程式：

```
 10 REM  OUTPUT TWO LINES OF TEXT TO A PRINTER
 40 REM  SELECT THE SERIAL I/O PORT
 50 PRINT "PR#1"
 60 PRINT "TO A SCREEN OR THE PRINTER AN APPLE WILL WRITE"
 70 PRINT "AND IN EACH CASE THE DATA GOES OUT BYTE BY BYTE"
 80 REM  DESELECT THE PRINTER
 90 PRINT "PR#0"
100 END
```

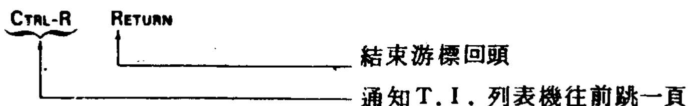
可以由程式控制的列表機

許多的列表機允許輸出的格式由程式來控制；藉著將一些適當的控制字加在輸出的文字資料內，您可以控制行的長度、字集的選擇、頁的長度以及其他列印時的特性。

許多普徧被使用的列表機都可以與APPLE Ⅱ連接；從程式設計的觀點看，APPLE Ⅱ對於任何種類的列表機一視同仁；例如，假設一台TEXAS INSTRUMENTS MODEL 810 的列表機與APPLE Ⅱ藉著在擴充接點 1 上的序列輸出介面卡連接，而在TEX-AX INSTRUMENTS 的手册上說明，我們可以依照下面的順序將一頁定爲六行：



除此之外，我們也可以依照下面的順序將列表機轉到下一頁的起始處(這被稱爲跳頁(FORM FEED))：



把前面那個列印的程式稍作修改，下面的 APPLESOFT 的程式

會把同樣兩行的資料列印 15 次，同時每六行佔用一頁的位置。下面的程式是假設 DOS 存在APPLE Ⅱ內。

```
10   REM    OUTPUT TEXT TO THE PRINTER
11   REM    USING 6 LINES PER PAGE AND 5 PAGES
20   REM    CREATE A CTRL-D CHARACTER
30 D$ = ""
40   REM    SELECT THE SERIAL I/O PORT
50   PRINT D$;"PR#1"
51   REM    SELECT 6 LINES PER PAGE
52   PRINT  CHR$ (27);"2"; CHR$ (6)
53   PRINT  CHR$ (12);: REM  PRINT FORM FEED TO POSITION FOR
 START OF OUTPUT
54   FOR I = 1 TO 15
60   PRINT "TO A SCREEN OR THE PRINTER AN APPLE WILL WRITE"
70   PRINT "AND IN EACH CASE THE DATA GOES OUT BYTE BY BYTE"
75   NEXT I
80   REM    DESELECT THE RRINTER
90   PRINT D$;"PR#0"
100   END
```

上面的這個程式中，行號 52 的敍述訂定每頁只能列印六行的規則；CHR$ (27) 這個函數就是表示 ESC 這個字，CHR$ (9) 則規定一頁除了印六行之外，另外空出三行做爲與下一頁的空隔位置，分號則被用來將所有的控制字連接起來構成必須的控制字串。

在行號 53 的敍述將跳頁的工作完成，CHR$ (12) 就是一個跳頁的控制字，而跟在後面的分號使得PRINT敍述的游標回頭功能消失，使不致於空印一行；因此，如果沒有這個分號，那麼第一頁將印出一行空白，而實際能印的位置只有五行，而在後面的每一頁都將印滿六行。

其他可以由程式控制的列表機都幾乎相同地使用一些不能印出的控制字做控制的工作。

上面這個程式由於使用了CHR$ 這個函數，因此不能被使用在整數BASIC語言中。您幾乎可以藉著按某個鍵或是幾個鍵產生任何的字串值，因此您可以使用一對引號將一個不能列印出的鍵包含在內代替CHR$ 的功能。例如，由三次的按鍵產生出的“ ESC ”的結果與CHR$ (27) 是完全相同的。您可以參考附錄 I 查看詳細的對照表。

將程式行列印出來

如果您在鍵盤上打入 LIST 命令，那麼存在APPLE Ⅱ記憶體內的任何程式都將被顯示在螢光幕上，而想要將程式印在紙上，您必須在輸入 LIST 命令之前先使用適當的PR# 敍述達成這個目的。假設列表機與位於第一個擴充接點上的介面卡連接，下面就是一些必須的步驟：

1. 確定想要被列印的程式存在APPLE Ⅱ的記憶體內。
2. 當游標位於一個空白行的位置上時，使用 PR#1 的命令選擇這個列表機，這時游標會回到這一行的起始處，但不會往下跳一行。
3. 這時打入 LIST 命令，這個命令將不會顯示在螢光幕上，但它會被印在列表機的紙上，而這個想要被印出來的程式也就被列印在紙上了。
4. 當整個程式被列印完畢之後，您就可以輸入 PR#0 的命令將輸出轉移回到螢光幕上，而這個命令同樣地仍然被印在列表機上而不是螢光幕上。

將資料存在磁帶上

我們在上一章中曾經學過了如何將程式存入磁帶內，以及如何將程式由磁帶抄入電腦中。在 APPLESOFT 中，您也可以將數值的及整數的矩陣存入磁帶內。

STORE 指令將矩陣的值存入磁帶內，而 RECALL 指令可以把這些值重新讀入電腦內；這些敍述即不能控制磁帶的轉動，也不會通

知程式使用者何時按下磁帶機的適當按鍵。下面的程式對於 **STORE** 及 **RECALL** 兩個指令做了一個示範；它將一些值放入一個數值矩陣內，並將這個矩陣存入磁帶，然後將這個矩陣所有的值變爲 $\emptyset$，最後則把存在磁帶上的值重新存回到記憶體內；這個矩陣的值分別在不同的程式執行時間內顯示，做爲矩陣值改變的一個記錄。

```
10  REM THIS PROGRAM DEMONSTRATES STORE AND RECALL
20  REM **************************************************
30  DIM A(10)
40  HOME
50  PRINT  TAB( 4);"STORED"; TAB( 13);"CLEARED"; TAB( 22);
 "RECALLED"
60  REM INITIALIZE ARRAY VALUES
70  FOR I = 1 TO 10:A(I) = I: NEXT I
80 T = 8: GOSUB 1000: REM DISPLAY VALUES TO BE STORED
90  VTAB 20: HTAB 1
100  PRINT "PLACE CASSETTE IN RECORDER. REWIND IT."
110  PRINT "PRESS THE 'RECORD' AND 'PLAY' BUTTONS,"
120  INPUT "AND ENTER 'GO' ";C$
130  IF C$ < > "GO" THEN  GOTO 90
140  STORE A
160  CLEAR : REM  SET ARRAY VALUES TO ZERO
170  VTAB 2:T = 18: GOSUB 1000: REM   DISPLAY CLEARED ARRAY
180  VTAB 20: HTAB 1
190  GOSUB 1100: REM   ERASE LAST INSTRUCTIONS
200  VTAB 20: HTAB 1
210  PRINT "REWIND TAPE.  PRESS 'PLAY' BUTTON,"
220  INPUT "AND ENTER 'GO' ";C$
230  IF C$ < > "GO" THEN  GOTO 200
240  RECALL A
260  VTAB 2:T = 28: GOSUB 1000: REM  DISPLAY RECALLED VALUES
265  VTAB 20: HTAB 1
270  GOSUB 1100: REM  ERASE LAST INSTRUCTIONS
280  VTAB 20: HTAB 1
290  PRINT "PRESS 'STOP' BUTTON."
300  END
990  REM +++++++++++++SUBROUTINE 1000+++++++++++++++
991  REM DISPLAY VALUES OF ARRAY A
1000  FOR I = 1 TO 10: HTAB T: PRINT A(I): NEXT I: RETURN
1090  REM +++++++++++++SUBROUTINE 1100++++++++++++++++++++
1091  REM  ERASE THREE DISPLAY LINES
1100  FOR I = 1 TO 119: PRINT " ";: NEXT I: RETURN
```

您可以將矩陣用某個名字存入磁帶內，而在呼叫它時使用另外一個名字，但是一般說來，您所存入矩陣的行列數必須與您所呼叫的矩陣的行列數相同；也有一些較爲複雜的例外情形我們將在第八章中做一個描

述。除非您想達到某些特殊的效果，否則在您使用 **RECALL** 命令呼叫矩陣時，所用到的行列數必須與您使用 **STORE** 命令存入的矩陣的行列數相同。

如何使程式更爲精簡有效 *

傳統上認爲，一個最好的程式也就是對於同樣的一個工作而言，它執行的速度最快而且使用的記憶體位置最少。當然這種雙重的目標有時也被中和一下，使得程式仍然是信賴度高的，而且容易設計、容易被瞭解以及在修改時不致發生太大的困難；如果您能夠花更多的時間在這些方面，那麼您可能獲得較孜孜於如何使程式更快、更節省記憶位置所得的益處更大。但是，如果您瞭解如何加快程式的執行速度並正確地使用記憶體位置，那麼您就可以很自然地設計出有效的程式而且在執行之後不再需要做太多的修正工作了。在這種情況下，我們將告訴您幾種設計程式的方法，使得您的程式執行速度加快而且節省地使用記憶體的位置。

有一些能使程式執行速度加快的技巧往往佔用較大的記憶體位置，而另一些方法則恰恰相反，您必須在速度與記憶體位置之間做一個抉擇。

快速地執行程式

避免使用常數（例如；ø、1øø、“ Y ”、“ENTER ”），而在程式中較早的時間就將值存入變數內，然後使用這個變數代替常數；這在您重複地在實數運算式中使用整數值時特別重要，因爲電腦將花費較多的時間把一個整常數的值轉換成實數值，而且當這種的轉換工作發生

* 這可以算是所謂程式格局 (programming style) 的論題，您可以參考“BASIC程式格局”，冼鏡光譯，儒林圖書公司出版——譯者。

在 FOR-NEXT 迴圈內，一個常用的副程式或是一個使用者自訂的函數中時，這種差別將更形顯著；這個方法的另一個好處就是它使您的程式更容易瞭解，而且當您想要改變這個常數值時，使用變數的方法將使設定敍述的改變容易得多。

盡量地使用在程式執行中較早被使用到的變數名稱。變數存在記憶體中的位置，都是依照先到先存的方式處理的，而在BASIC語言中，存在前面的變數將較存在後面的變數更快地被尋找到。

電腦執行BASIC語言時，當它碰到一個轉移到某一行號的敍述，它將從這個程式的起始行號開始順序地尋找，直到找到這個行號時爲止；因此，我們可以清楚地看出來，對於較小數目的行號，電腦能夠較快地找到它的位置，所以將最常被使用到的副程式給予它最低的行號。

在 APPLESOFT 中，不要在 NEXT 敍述中加上脚註變數，因爲這麼做將使電腦不必去求證您所使用的脚註是否正確。

如何節省記憶體位置

對於同樣的邏輯使用副程式，而避免重複地設計同樣的程式，這同樣地使您的程式更易於被瞭解、更可靠以及在改變時更爲簡易。

使用矩陣的第零號元素（例如，X（ø）、B（ø））。

常數通常較變數佔用更多的文字數，所以將常數存入變數內，替換常數的使用。

由於每一個程式行多佔用五個 BYTE 的位置；所以，將不祇一個的敍述放入同一程式行中，將比較節省記憶位置；但是需要注意的是，這種混合敍述的程式行往往是較難編修及瞭解的；因此，在使用混合敍述的方式時必須仔細考慮妥當。

明智地使用 REM 敍述，儘量縮短說明的長度；但您必須小心的是，說明愈少，將來您重新研究程式所遭遇的困難將更多。

不要過份地使用變數，因爲每一個變數都佔用一定數量的記憶體位

置，縱使您只使用一次也是一樣。所以建立一套設定變數名稱的系統，並把使用在FOR-NEXT迴圈、暫時性變數等等只被使用一次的變數也包含在內，但是也不要過份地使用它們。如果您依照變數所使用的目的給名稱，那麼將使您的程式更易於瞭解，對於獨一的變數建立起標準的名稱（例如，CN$ 代表客戶名稱），對於一組的變數也同樣地建立起標準的名稱（例如，所有只被使用在一個地方的變數都用X開頭）。

使用INPUT 敍述以及資料檔（DATA FILE）——您可以參考第五章磁碟機的部份——代替使用設定敍述及 DATA 敍述。

在 APPLESOFT 中，使用整數矩陣代替實數矩陣，因為每一個整數矩陣的值只佔用兩個 BYTE 的位置，而實數矩陣的值則佔用五個 BYTE 。在您的程式中週期性地使用 FRE 功能將記憶體中儲存字串的位置清除掉。

除 錯

當您設計了一個新的程式之後，往往它並不能如您所希望地達成工作，即使在BASIC語法方面沒有錯誤，在邏輯上却往往會發生不可預知的錯誤，而這任何一種的錯誤就被稱為障礙（ BUG ），至於消除程式錯誤的過程就被稱為除錯（DEBUGGING）。下面有許多的方法使您可以做除錯的工作。

通常在您設計程式之前，您最好能夠花一些時間做較詳盡的計劃，不要對您所要做的工作只是有個大概的意念時就坐在電腦的前面直接設計程式，這樣做往往會浪費您更多的時間。如果設計程式對您來說是一件剛開始不久的事，那麼您最好多參考一些設計程式的書籍以便增加您的能力。

如果您設計的程式並不能如您所期望地工作，而您實在想不出原因，那麼您可以參考使用下面的一些BASIC 敍述，它們將對您程式的除

錯工作提供極大的幫助。

PRINT敍述

PRINT 敍述是一個非常有用的除錯工具，想必是令您驚異的事吧！您可以將 **PRINT** 敍述放在程式中的某幾個關鍵位置，在程式執行到某個程度時，顯示出執行成功的訊息，並把某些變數的值顯示出來，這就使得您可以追蹤程式執行的流程以及檢查某些變數在執行中的值了。

TRACE敍述

TRACE 敍述的功能與它的名字相符，它藉著顯示被執行敍述的行號而追蹤整個程式的執行流程。想要瞭解它的功能，您可以輸入下面的程式，再打入 **TRACE** ，然後執行這個程式：

```
100  PRINT "ENTER A NUMBER FROM 1 TO 5 (6 TO END)";
110  INPUT N
120  IF N = 1 THEN  PRINT "UNO";
130  IF N = 2 THEN  PRINT "DOS";
140  IF N = 3 THEN  PRINT "TRES";
150  IF N = 4 THEN  PRINT "CUATRO";
160  IF N = 5 THEN  PRINT "CINCO";
170  IF N > 5 THEN  GOTO 100
180  FOR I = 1 TO N
190  PRINT " *";: REM  PRINT N ASTERISKS
200  NEXT I
210  CALL  - 936: REM  CLEAR SCREEN
220  GOTO 100
```

您可以使用**NO TRACE** 取消 **TRACE** 的功能，而返回到正常的狀況。

DSP敍述

下面有一個只能在整數BASIC敍述中被使用的除錯敍述；那就是DSP 敍述。這裏有一個例子：

```
>10 DSP COUNT
```

只要這個特別的敍述被執行之後，APPLE Ⅱ將在 COUNT 這個變數的值改變時通知您，而且告知您是在那一個行號時這個值改變的。由於DSP 敍述會被 RUN 命令取消，因此您必須使用 GOTO 敍述開始程式的執行，或是將 DSP 敍述放在您的程式中。

您也可以使用 NO DSP 敍述取消DSP 的功能；下面就是一個例子：

```
>300 NODSP NAME$
```

只要這個敍述被執行了之後，APPLE Ⅱ將不再通知您變數NAME $ 的變化情形。

您可以在上面我們使用 TRACE 敍述的例子中，試著加入下面的程式行，以便瞭解DSP 的效用。雖然我們可以同時使用 TRACE 及DSP 敍述，但是在執行下面的程式時，您最好將 TRACE 敍述取消，以便更加瞭解DSP 敍述的功用。

```
10 DSPN
20 DSPI
100  PRINT "ENTER A NUMBER FROM 1 TO 5 (6 TO END)";
110  INPUT N
120  IF N = 1 THEN  PRINT "UNO";
130  IF N = 2 THEN  PRINT "DOS";
140  IF N = 3 THEN  PRINT "TRES";
150  IF N = 4 THEN  PRINT "CUATRO";
160  IF N = 5 THEN  PRINT "CINCO";
170  IF N > 5 THEN  GOTO 100
180  FOR I = 1 TO N
190  PRINT " *";: REM  PRINT N ASTERISKS
200  NEXT I
210  CALL  - 936: REM  CLEAR SCREEN
215 NODSPN
220 . GOTO 100
```

立即式與間接式的限制

許多的 BASIC 敍述可以被使用在立即式或是間接式的情況中，但是有一些的敍述却只能在某個方式下使用。表4－2中列出了在整數 BASIC 語言中的一些被限制使用的敍述，表4－3中則列出了在 APPLESOFT 中被限制使用的敍述。

表4－2 整數 BASIC 中受到立即式及間接式執行限制的敍述

只能在間接式執行	只能在立即式執行
END FOR GOSUB INPUT NEXT RETURN	AUTO CLR CON DEL HIMEM: LOAD LOMEM: MAN NEW RUN SAVE

表4－3 APPLESOFT 中只能在間接式情況下執行的敍述

只能在間接式執行
DATA DEF FN GET INPUT ON ERR GOTO RESUME

5

DISK Ⅱ 磁碟機



在一個電腦系統中，磁碟機是最重要的設備之一，磁碟機允許您處理一大筆資料中的任何一項；**APPLE** 的 **DISK Ⅱ**磁碟機可以在一片磁碟片上儲存超過143,000個字的資料，而這幾乎是48K **RAM** 的三倍，而且當電腦的電源被切斷之後，儲存在磁片上的資料仍然存在。

磁碟機的種類

磁碟機與磁帶機都是利用磁性的方式儲存資料的，它們之間最大的差別就是磁碟如同唱片一般是圓的，而且像唱片一般地轉動；在磁碟機的內部有一個**磁頭**(**HEAD**)，磁碟機就利用它來讀和寫資料，電腦可以把磁頭移到磁碟面的任何位置，而這種能力就被稱為**隨機處理** (**RAN-DOM ACCESS**) ，因此，磁碟機也就是一種隨機處理儲存裝置(**RANDOM ACCESS STORAGE DEVICE**) 。磁碟機的作業由一個特殊的程式來控制，這個程式就被稱為磁碟作業系統 (**DISK OPERATING SYSTEM**) 或簡稱為**DOS**。下面有幾種不同種類

的磁碟機。

硬性磁碟機

硬性磁碟（HARD DISK）是不能彎曲的而且外面加上了一層磁性的物質，硬性磁碟通常可以儲存五到十個MBYTE的資料（1MBYTE＝1,000,000BYTE）；大部份的硬性磁碟都可以拿出來的，也就是說，磁碟與磁碟機是分開的，所以您可以更換磁碟。每一個硬性磁碟大約值美金 150 元左右，而硬性磁碟機則在 3,000 至 10,000元之間。圖5－1 就是一個典型的硬性磁碟機系統。

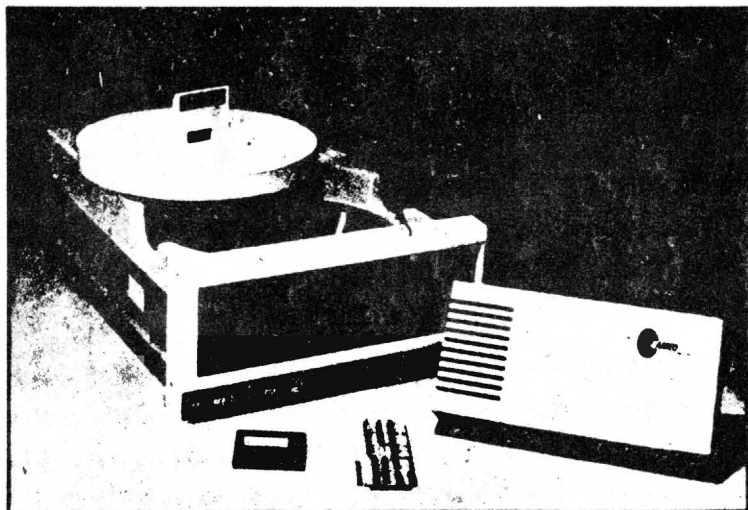


圖5－1　典型的硬性磁碟系統

溫徹斯特磁碟機

圖5－2 就是一個溫徹斯特磁碟機（WINCHESTER DISK PICTURED），它使用一種特別的技術使得磁碟的容量可以增大六至十倍，但是它必需絕對地與灰塵隔絕－即使是一點點的香煙灰也不可以；由於它必需絕對的乾淨，所以溫徹斯特磁碟被密封在磁碟機內而且不能被更換，它的價格大約在美金2,000與8,000之間。

軟性磁碟片

軟性磁碟片是最常被使用的磁碟，它是一個圓型的磁片被塑膠的外

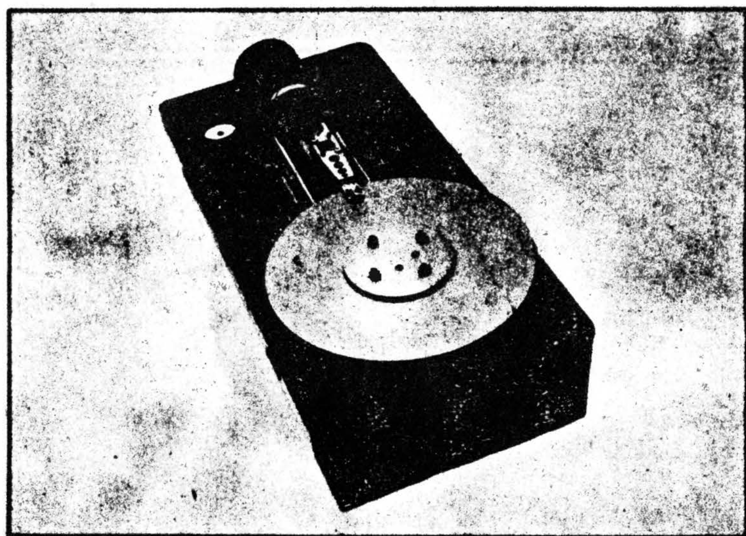


圖5－2 溫徹斯特磁碟機

殼包在裏面；外殼就是用來保護磁碟片的，磁片在外殼內可以自由的轉動，在封套的一些開孔的地方就是容許磁頭處理磁片表面的所在，同時也提供一個使磁碟機能夠轉動及停止磁片的區域。磁碟片絕對不能從封套內移開。圖 5—3 就是一個磁片沒有封套的模樣。

軟性磁碟片有兩種不同的型式：直徑 8 英吋的及直徑 5¼ 英吋的，這兩種不同的型式都顯示在圖 5—4 中。**APPLE** 的 **DISK Ⅱ** 磁碟機使用的 5¼英吋的軟性磁片，**DISK Ⅱ** 可以在每片軟性磁片上儲存 143, 360 個 Byte 的資料。

資料如何儲存在磁碟上

您應該要對不同層面的磁碟儲存過程非常熟悉，資料儲存在磁碟上

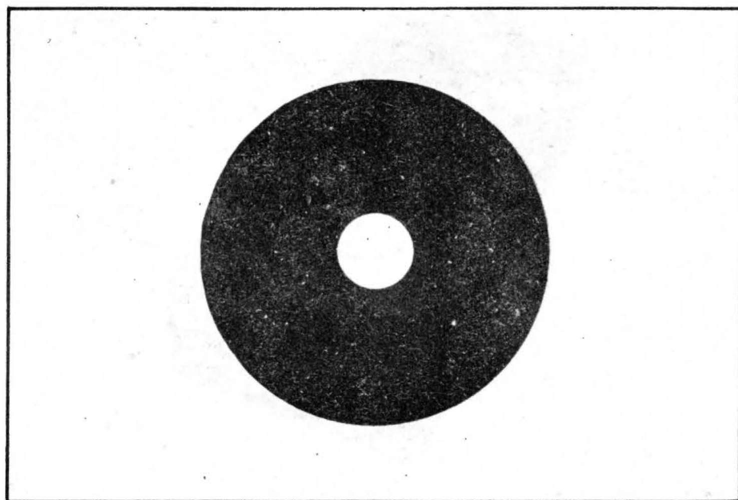


圖 5 — 3　沒有保護封套的磁碟片

是一連串互相配合的動作所造成的結果。

磁 軌

為了要使處於 143;360 個 byte(數元組)中的一個 byte 能夠迅速被找到,**APPLE**的**DOS**將磁片分為35條磁軌(**TRACKS**),分別編號為 0,1,2,……34;磁軌與唱片上的槽相似,只是您看不到它罷了,而且它們並不相連;也就是說,它們是一組同心圓,而且每個都包含在另一個的裏面,就如同圖 5-5 所顯示的一般。

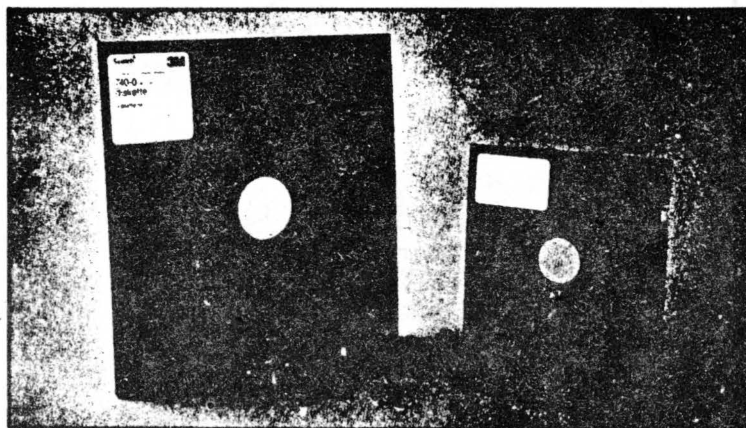


圖5-4 8英吋及$5\frac{1}{4}$英吋磁碟片

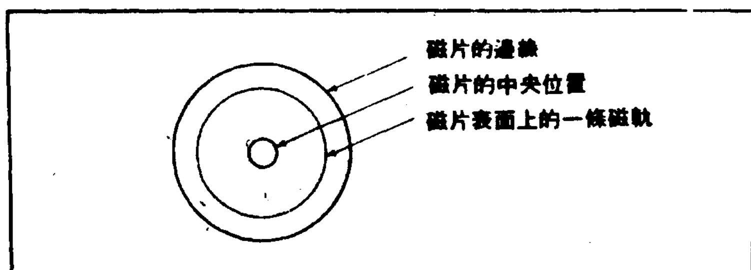


圖5－5　磁片上的磁軌

磁區

磁軌

一個磁區內儲存256個數元組的資料

圖5－6　磁片的表面

磁 區

為了要使某一個 byte 更快地被找到，DOS 將磁軌又分為16個磁區（SECTOR），就如圖 5–6 所顯示的一般。每一個磁區擁有 256 個byte，對於某一個byte 而言，如果將它限制在某個磁軌及磁區內，那麼DOS 只需要找 256 個byte 就可以了；因此，處理某一個 byte 或一串的byte 就幾乎是非常快速的事了，而磁碟機的眞正效用才能發揮出來。

標示磁軌及磁區的位置

在磁片上尋找某一個磁軌是很簡單的：磁碟機將磁頭移到所要找的磁軌上就可以了，就如同您在唱片上將唱頭移到您所要聽的那一首歌上一般。

但是尋找某個磁區就比較困難了，有兩種常用的方法將磁片上的磁區標示出來，兩種方法都是利用一個被稱為脚註孔（INDEX HOLE）的磁片上的孔來作這個工作。大多數的磁片上的這個脚註孔就在封套中央的一個大孔的旁邊，當磁片轉動時，磁片本身具有的孔就通過封套上的這個孔，而在磁碟機內的一個光源當磁片的孔經過時就會被送到一個感應器，電腦這時就會感應到這些光，同時依照這些資料而計算磁區的位置。

下面就有兩種方法可以用來尋找磁區，分別被稱為硬磁區（HARD SECTORING）法及軟磁區（SOFT SECTORING）法。

硬磁區

硬磁區的磁片有許多個孔，如同5–7 一般。每一孔指出個別磁區的位置，一個特別的孔標示出第一個磁區的位置，電腦尋找磁區就是利用計

算跟在第一個磁區後的孔的數目而達成的。DISK Ⅱ並不是使用這個方法來尋找磁區。

軟磁區

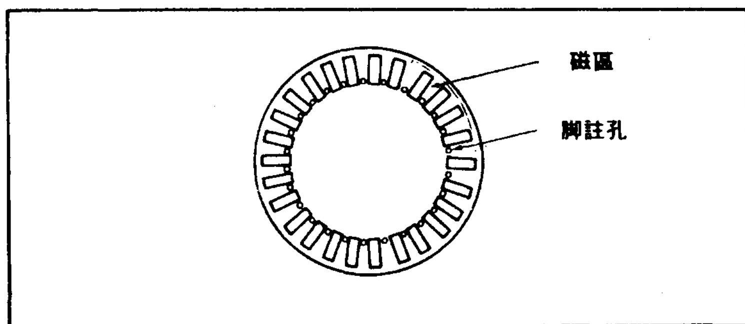


圖5－7　具有硬磁區的磁片

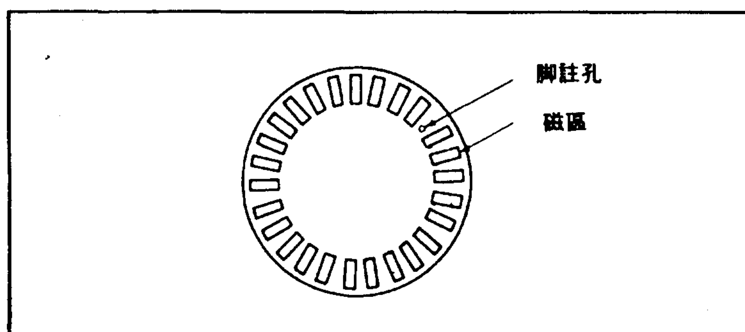


圖5－8　具有軟磁區的磁片

具有軟磁區的磁碟只有一個脚註孔，就如圖5–8顯示的一般，它只標示第一個磁區，至於尋找其他的磁區則是用計算磁碟機轉動的時間而達成的。

APPLE的DISK Ⅱ並不管計算時間的孔，它的計時方法是電子而非物理的，這就是說您可以在**DISK** Ⅱ上使用軟磁區的磁片或是硬磁區的磁片，它們的結果都是一樣的。

磁碟片的僅讀保護

在磁片封套的一邊有一個缺口，這個缺口就是用來控制資料寫入磁片的裝置。在8英吋的磁片上，這個缺口被稱爲僅讀（**WRITE PRO-TERT**）缺口，因爲當缺口存在時，電腦將不會把資料寫在磁片上；在5¼英吋的磁片上，這個缺口就被稱爲可寫（**WRITE ENABLE**）缺口，因爲只要這個缺口存在，電腦就可以將資料記錄在磁片上。有一些磁片，像"**SYSTEM MASTER DISKETTE**"就是永遠被保護住的，因爲它沒有任何的缺口，想要將5¼英吋的磁片變爲僅讀磁片，只需要將一小塊的膠帶貼在缺口上就可以了。圖5–9就是一個示範。

磁碟作業系統

所有與磁碟相關的作業都由一個特殊的程式控制，而這個程式就稱爲磁碟作業系統（**DISK OPERATING SYSTEM**）或是**DOS**。**BASIC**將任何對磁碟作業的要求都傳送到**DOS**，而**DOS**將結果傳回給**BASIC**。

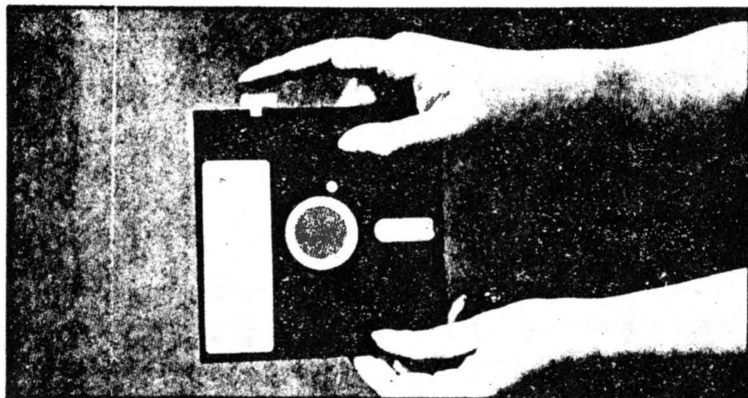


圖 5－9 僅讀的5¼英吋的磁片

DOS的型式

DOS 有幾種不同型式，DOS 3.3 是最近的一種，在本章中我們所要描述的也是它；至於DOS3.3與DOS3.2.1 最大的不同就是磁片上磁區數目的差異，DOS 3.2.1有 13 個磁區，而DOS3.3 則分爲16個磁區。APPLE 電腦公司有一個特殊的程式可以把DOS3.2.1的磁片轉換成DOS3.3的磁片。

設定磁片的格式

在APPLE Ⅱ能夠使用一個磁片前，這磁片必須先被設定格式(INITIALIZED)，設定格式的工作是把磁片上的所有資料全部消除，並把

DOS 抄到在∅，1及2號磁軌上。至於DISK Ⅱ設定格式的指令我們將在稍後介紹。

磁碟資料檔

資料是以檔案(FILE)的方式存在磁片上的，一個檔可以是任何的長度，只要是能配合磁片本身的結構就可以了；每一個檔都有一個名字，一個檔可以儲存字的資料，一個程式或是APPLE Ⅱ繪圖顯示的影像，這些不同型式的檔都將在稍後詳細描述。

磁片的索引檔

在磁片上每一個檔的名稱都被存在這個磁片的索引檔(DIRECT-ORY)中，索引檔都放在該磁片的第17號磁軌中。索引檔的第一個輸入資料位在第15個磁區，而最後一個輸入資料則位於第一個磁區。一個索引檔可以記錄到104個檔的名稱及相關的資料。

與檔的名稱同時被儲存的是一個能夠指出這一個檔中資料型式，該檔案佔磁區的數目以及存有這個檔磁軌/磁區表的磁區位置的碼。

磁軌/磁區表

磁軌/磁區表(TRACK/SECTOR LIST)由一組一組的byte 組成，這些 byte 存有被這個檔使用到的磁軌及磁區的位置，每一組就被稱爲一個聯結(LINK)；磁軌/磁區表中的第一個聯結裏面存著的就是被這個表所使用到的下一個磁區的位置。這個表可以使用任何數目的磁區；第二個聯結存的就是這個檔所使用的第一個磁區的位置，第三個聯結則存放下一個磁區的位置，如此繼續地依照這種方式儲存下去，直到這個表結束(也就是被標示爲第零個聯結)時爲止。

磁碟儲存過程的回顧

資料存入或由磁碟中取出，都是由DOS 控制的，當您將資料寫入一個檔中時，有幾件事情將會發生：

1. DOS 到磁碟索引檔中尋找這個檔的名稱（在您的程式中您必須給每一個檔一個名字，這一點我們將在稍後描述）。
2. 如果這個檔的名字找到了，DOS 就會從適當的磁區內讀出 256 個byte的資料，並把它們存在記憶體的一個叫做緩衝區（BUFF－ER）的區域內，如果名字沒有被找到，或是這個檔的這個磁區的資料已經被寫進去了。那麼DOS將把緩衝區內全部放滿了零。
3. 最多的 256 個byte 的資料可以被寫進緩衝區，如果資料不足 256 個byte ，那麼寫進緩衝區內時，以前所遺留的資料仍然存在。
4. 如果將剛好256個byte的資料被寫入緩衝區內時，那麼緩衝區內的資料被寫回適當的磁區中。
5. 第三及第四步驟將重複地被執行，直到所有要寫入這個檔的資料都已經被寫到磁片上為止。
6. 所有的資料都已被寫入磁片上之後，DOS 將把磁軌／磁區表及索引檔內的記錄做一個修正。

需要注意的是，除非您要寫入檔內的資料的byte數目可以被 256 整除，否則最後一個資料區（BLOCK）將不會放入緩衝區內；也就是說，步驟 4 到步驟 6 將不被執行了，因此就有一個DISK Ⅱ的命令叫做CLOSE的，強迫存在緩衝區內的資料寫入檔內，然後修正磁軌／磁區表及索引檔的內容，這個過程就被稱為關閉（CLOSING）這個檔。如果您在把資料寫入檔後而不能將這個檔關閉，那麼很可能您將失去您的資料，因此記得在您結束處理這個檔之後，務必要將它關閉。CLOSE命令我

們將在稍後做更詳盡的描述。

磁碟的毀損

磁碟發生錯誤中的最嚴重一項就是磁碟毀損（DISK CRASH），這有兩種不同的型式：硬體的毀損（HARD CRASH 及軟體的毀損（SOFT CRASH）。

硬體的毀損發生在磁碟的表面受到損害時，或者是有結構上的污點，例如一個小的污點或是一小塊灰塵。硬體的毀損往往造成讀寫頭的損壞，而這個受損的讀寫頭就會造成更多磁片的損害。因此，當您使用磁片時需要特別地小心。

軟體的毀損發生在存有索引檔的磁軌被不正確的資料所覆蓋時，它最常發生在下面的情形：當有一個或是多個的檔被寫入磁片但是並沒有將它們關閉，然後有另外一個磁片被放入磁碟機內，接著您才將前面那個磁片內的檔關閉。為了要避免這些混亂您必須多吸收些經驗。

將DOS抄入記憶體內

為了要使APPLE能夠瞭解任何磁碟的命令，我們必須將DOS 存入記憶體內。如果您有許多時間，那麼您可以經由鍵盤將DOS 輸入到記憶體內，但是這裏有一個簡單的方法，那就是被稱為BOOTING磁碟；BOOTING磁碟，或是BOOTING DOS 就是從磁片上將DOS讀出並把它存入記憶體內。

如何將DOS抄入記憶體內

下面有許多種不同的方法，完全依照您所擁有的APPLE Ⅱ的型

式及所使用的語言而將DOS 存入記憶體內。每一個方法都假設磁碟機是與插在擴充接點6上的控制卡的第一個接頭相連接的。

將"SYSTEM MASTER DISKETTE"磁片放入磁碟機內並把門關上；下一步所要做的事完全依照您所使用的APPLE系統而定了，當一個成功的BOOT 動作完成後，顯示螢光幕將如圖2–5中所顯示的其中一個。

自動啓動系統

如果您擁有自動啓動系統，那麼BOOT 的動作將是最簡單不過的了，就如它的名字所顯示的，BOOTING將是全自動的；但是想要自動啓動，您必須擁有自動監督系統。您可以將DISK Ⅱ與主機連接後打開開關，這時若是DISK Ⅱ上的紅燈亮了，而且磁碟機發出轉動及摩擦的聲音，那麼您的APPLE Ⅱ就擁有自動監督系統。

若要由自動監督系統內將DOS 存入記憶體，那只要簡單的將APPLE Ⅱ的開關打開就可以了。

由監督系統中將DOS存入記憶體內

當監督系統的系統標示符號（＊）出現在螢光幕上時，APPLE Ⅱ的監督系統編譯程式就在等待著接受命令了，這裏有幾種不同的方法將DOS 由監督系統存入記憶體內。

跳出監督系統的Booting

打入C字，後面跟著磁碟機所連接的擴充接點的數目（通常是6），然後打入兩個ø及G字。這整個的命令就如下面的一般：

```
*C600G
```

C600 就是能夠將連接在第六個擴充接點上磁碟機內的資料存入記憶體內的程式的位置，G就是能把控制轉移到這個程式的命令。現在您就可以按下RETURN 鍵了。這時磁碟機的燈就會閃亮而且發生轉動及摩擦的聲音了。

使用 CTRL-K及 CTRL-P做Booting

您也可以使用CTRL-K或CTRL-P將DOS 在監督系統內存入記憶體中；在監督系統下使用這兩個命令，您只需打入擴充接點的數目（正常的情況下就是6）然後打入CTRL-K或CTRL-P，這兩個命令都不會顯示在螢光幕上的。

在打入命令之後，您就只需要按下RETURN 鍵就可以了。

由整數 BASIC或Applesoft中做Booting

在整數BASIC或APPLESOFT中可以使用同樣的命令做BOOT-ING的動作。

在BASIC語言中使用PR# 及IN#做Booting

在BASIC語言的系統提示（在整數BASIC是>，在APPLE-SOFT爲]）後打入PR或IN，然後打入一個#號（#），最後打入與磁碟機相連接的擴充接點的數目。這個命令就將如下面的兩個命令中的一個一般。

PR#6
或
IN#6

現在您可以按下RTEURN 鍵了。

擁有語言系統時的BOOTING

如果您擁有語言系統卡並把它插在您的機器上，那麼上面所提的**BOOTING** 動作也許必須稍作修改。當您的系統是**DOS** 3.2.1,3.2 或是更低的版次，那麼就必須使用兩片磁片才能做**BOOTING** 了。但是**DOS** 3.3只需要與上面相同的步驟就可以了。

想要將**DOS** 3.2.1 或**DOS** 3.2 存入記憶體（當語言系統存在時），首先插入標有"**BASICS：INTEGER AND APPLESOFT Ⅱ**"的磁片，然後依然前面描述的**BOOTING** 動作一般作業，在這個作業成功了之後，螢光幕將顯示：

INSERT BASIC DISK AND PRESS RETURN

這是使人有些迷惑的顯示，但是事實上您只需要將**DOS** 磁片（"**SYSTEM MASTER DISKETTE**"也可以，或是任何一片被設定格式後的磁片）插入，然後您就可以按下**RETURN** 鍵，這時**BOOTING** 的動作就會正確地進行了，而您也將看到如同圖2–5 中的一個圖形的情形。

開始學習磁碟命令

APPLE Ⅱ的**DOS** 解譯及執行磁碟命令，許多的命令需要或是允許額外的元素以便定義執行時的操作。下面將描述一些基本的命令。

CATALOG命令

CATALOG（目錄）命令將磁碟上索引檔內所有檔的名字輸出到

輸出設備上，一般來說就是輸出到螢光幕上。

第一個顯現的是磁碟上的磁碟片號碼（VOLUME NUMBER）。

對於每一個在磁碟上的檔而言，CATALOG 將這個檔中資料的型式顯現出來，這個檔是否被鎖住（LOCKED），這個檔所佔用的磁區數目以及這個檔的名字都會顯示在螢光幕上。一個典型的目錄就如圖5－10所顯示的一般。

檔的型式

檔內資料的型式是使用一個字來表示，它的位置就在目錄最左邊的那一行。不同的碼代表不同的資料型式，就如同表5－1所顯示的一般。

被鎖住的檔

被鎖住的檔就是不能被改變或消除掉的檔，目錄中使用一個（＊）

```
*I 002 HELLO
*I 053 APPLE-TREK
*I 018 ANIMALS
*B 009 UPDATE 3.2.1
*I 014 COPY
*I 009 COLOR DEMO
*I 053 BRICK OUT
*I 026 SPACE WAR
*I 050 THE INFINITE NO. OF MONKEYS
*I 051 COLOR SKETCH
*I 053 SUPERMATH
*I 026 APPLEVISION
*I 017 BIORHYTHM
*I 027 PINBALL
```

圖5－10 典型的磁碟目錄

表5－1 磁碟檔型式碼

碼	意 義
A	APPLESOFT程式
B	二進位映設檔
I	整數BASIC程式
T	文字資料檔
R	可被執行的二進位檔

號放在型式碼的前面用來表示被鎖住的檔，如果型式碼前面沒有（＊）號，那麼就表示這個檔沒有被鎖住。至於被鎖住的檔我們將在本章稍後的地方討論。

佔用磁區的數目

這個檔所佔用的磁區的數目是由一個具有三個數字的數目來表示的，最小的檔（空的）佔用一個磁區，如果一個檔佔用了超過255個的磁區，那麼這個數目將從頭由∅再開始計算，但是這並不影響這個檔的眞正佔用位置。

檔的名稱

APPLE Ⅱ的DOS 需要您在使用檔時指出它的名字，下面就是一些您設定檔名字時所必須遵守的規則。

1. 檔的名字的長度必須在1到30個字之間，超出的字將不被承認。

2. 檔的名字必須由字母開始。
3. 任何您可以由鍵盤上打入的字都可以做爲檔名字的一部份，但是逗號是一個例外。

您也可以使用不顯示在螢光幕上的字（例如與CTRL 鍵合用時）做爲檔的名字的一部份，但它們將不會顯示在目錄表上，如果您想要防止他人知道您的檔名，這不失爲一個好方法（但您不要忘了所使用的不顯現的字）。

使用CATALOG命令

使用CATALOG命令，您只需打入這個字（假設DOS 已經被存在記憶體內了）就可以了。

```
CATALOG
```

輸入這個命令的結果應該與圖5-10 所顯現的相似。

如果CATALOG 命令所顯示的行數超過了20行，那麼電腦將顯現出前面的21行，然後等待您按下任何一個鍵（除了RESET，CTRL，及SHIFT鍵之外）然後將下一個21行顯示出來；這個暫時的等待使得您有時間去瞭解所有檔的名字。

LOAD命令

LOAD 命令由磁片上讀入一個程式檔放在記憶體內，您必須指出這個檔的名稱。下面這個例子將COLOR DEMOSOFT 這個程式存入記憶體內：

```
LOAD COLOR DEMOSOFT
```

如果您所指出的檔的名字並不存在磁碟索引檔內，那麼您將得到**FILE NOT FOUND** 的錯誤訊息。

如果這個檔在這片磁碟上，**DOS** 將會檢查這個檔中的資料型式，如果它不是一個程式檔，那麼您將得到**FILE TYPE MISMATCH**的錯誤訊息。

假設一切的情況都是令人滿意的，那麼**LOAD** 命令將把存在**RAM**中的程式消除掉，然後將這個檔的程式複製到**RAM** 內，在系統提示及游標出現在螢光幕上之後，您就可以列印，修改或執行這個新被存入的程式了。

RUN命令

在您使用過**LOAD** 命令之後，往往您就會使用**RUN** 命令，而您可以將這兩個步驟的過程簡化成一步就可以了，您在**RUN** 命令之後加上您所要執行的檔的名字，使得**LOAD** 命令變成爲隱藏的步驟，因爲一個檔在被執行之前必須先被存入記憶體內。下面就是幾個例子。

```
RUN PROGRAM 2
RUN SPOT RUN
RUN COLOR DEMOSOFT
```

指出磁碟機的號碼

許多**DOS** 的命令允許您指出您所要使用的磁碟機的號碼，您可以使用兩個元素來達成選擇某一個磁碟機目的：磁碟機元素及擴充接點元素。

由於一片磁碟機介面卡可以與兩個磁碟機相連，因此您在選擇磁碟機時，只需要在磁碟命令中加上一個逗號及**D**1 或**D**2 ：

```
LOAD UP,D2
RUN AROUND,D1
CATALOG,D2
```

在您指定了某一個磁碟機是您所要使用的之後，在您以後使用到的磁碟命令中，這個磁碟機就被設定爲處理的對象，直到您再指定其他的磁碟機後爲止；這個設定的磁碟機也就是最近的一個磁碟命令中所指定的那一個，如果您在命令中都沒有指定任何磁碟機，那麼第一個磁碟機就是被設定的磁碟機。

擴充接點的設定

DISK Ⅱ的控制介面卡是插在APPLE Ⅱ機器裏面的擴充接點上的，APPLE Ⅱ本身具有八個擴充接點，但是第0個擴充接點上並不能插入DISK Ⅱ控制介面卡，因此只剩下七個擴充點可供DISK Ⅱ控制介面卡佔用；由於每個控制介面卡可以與兩個磁碟機相連，所以APPLE Ⅱ最多可以與14個磁碟機相連接。

如果您的主機與超過兩個的磁碟機相連接在一起，那麼您不能只是用D3，D4…… 等代號來代表它們，您必須使用其他的元素選擇適當的控制卡，而這個元素就是擴充接點，您使用它指出您所要使用的磁碟機是與位於那一個擴充接點上的控制卡連接的。

使用擴充接點做爲元素，您在磁碟命令中加入一個逗號，S字以及擴充接點的數目（1－7），就像下面的例子一般。

```
CATALOG,S5
LOAD TRUCKS,S6
RUN OVER,S3
```

這被指定的擴充接點在DOS 被存入記憶體內時，就變成爲設定的擴充接點，直到其他的擴充接點又被指定時爲止。

在您使用了一個擴充接點做為磁碟命令中的元素之後，其餘的磁碟命令就被自動設定在同樣的擴充接點上，直到您再指定其他的擴充接點時為止。

您也可以同時使用磁碟機及擴充接點做為磁碟命令之元素，您可以指出1到7的擴充接點數目以及一個1或2的磁碟機數目，而使用任何一個與APPLEⅡ連接的磁碟機，它們在磁碟命令中的順序並沒有規定。下面的這兩個命令是相同的：

```
CATALOG,D2,S5
CATALOG,S5,D2
```

這兩個命令都將把接在第五個擴充接點上的第二個磁碟的目錄顯示出來。

選擇擴充接點時的問題

如果您所選擇的擴充接點上並沒有控制卡，那麼電腦將被鎖住，電腦等待著並不存在的控制卡給它一個準備好了的訊號，您可以在這種情形下按 RESET 鍵，使您的程式能重新動作，如果這時您得到的是BASIC 的系統標示（>或］），那麼一切都沒有問題，如果您得到的是監督系統的系統標示（＊）您就必須打入3DØG並按下RETURN，如果這樣做也不能成功，那麼由於您必須將DOS 存入記憶體內，您的程式也將會被毀掉了。

磁碟片的指定

磁碟片（VOLUME）是您在使用磁碟命令時可以與磁碟機、擴充接點等同時被使用的元素（除了CATALOG命令之外），它允許您確定在您所指出的磁碟機內的磁片是否是您想要的那一個。

CATALOG不管磁碟片元素，當CATALOG命令將磁片中的目錄列出時，磁碟片的號碼就是第一個被列所的元素。

使用磁碟片做爲元素時，您只需加入一個逗號，V字然後加入磁碟片的號碼，就如以下一段：

```
LOAD ZONE,V191
```

如果您所指出的磁碟片號碼與磁片上的號碼不同時，DOS 將會傳回**VOLUME MISMATCH** 的錯誤訊息。

這個號碼必須在1與254之間，如果您沒有指出它，或是您使用0做爲磁碟片號碼，那麼DOS 就不理會這個元素。

您也可以把磁碟片號碼與擴充接點，磁碟機等元素相連接在磁碟命令中使用，下面就是一些例子：

```
LOAD CARGO,D2,V24
RUN PAYROLL,S6,V111
```

更多的DISK Ⅱ 命令

您現在已經知道了如何使用**CATALOG** 命令觀看磁片上的目錄，也知道如何將程式存入記憶體內並執行它，現在您已經可以將您的程式使用**INIT, SAVE, DELETE, LOCK, RENAME**及**VERIFY**命令存在磁片上了。

INIT命令

在您將資料或程式寫入磁片之前，磁碟片必須先被設定格式；當一個磁片被設定格式的時候，任何存在磁片上的資料或程式都將被消除掉，所以您在設定磁片格式時，必須確定磁片上面的訊息都是您所不需要再保存的了。

當您使用 **INIT** 命令時，它會把目前記憶體內的任何程式存入磁片內，而這個程式也就變成歡迎程式（**GREETING PROGRAM**），當您做**BOOT** 的工作時，這個程式就自動地被執行，歡迎程式可以是簡單的或是複雜的完全由您決定。例如，假設您有一個存有郵遞資料（**MAILING LIST**）的磁片，那麼您就可以使用這個郵遞資料程式做為歡迎程式，當您將磁片放入磁碟機內，並做**BOOT**的動作時，這個程式就自動地被執行了；另外一個例子就是歡迎程式中只包含了**NEW**及**END** 敍述，那麼每一次這個磁片被**BOOT** 的時候，您就會得到**BASIC**的系統提示（＞或〕）。

一個設計良好的歡迎程式應該告訴您一些有關於這個磁片的資料。下面就是一個典型的歡迎程式：

```
100  TEXT
200  CALL  - 936
300  PRINT "THIS IS MY FIRST DISKETTE."
400  PRINT
500  PRINT "INITIALIZED 2/16/81"
600  PRINT
700  PRINT "ON A 48K SYSTEM USING DOS 3.3"
800  END
```

當您使用 **INIT** 命令時，歡迎程式所在的檔的名稱必須被指出，而您必須要能確定在這個磁片上永遠都有這個名字所代表的程式，如果您將這個歡迎程式消除掉了（我們將在稍後描述如何做），那麼每次在您**BOOT** 這個程式時，您都將得到**FILE NOT FOUND** 的錯誤訊息，而想要消除這個錯誤訊息的方法就是在歡迎程式的名下實際放入一個程式，但是由於您可能忘記了歡迎程式的名字，這個工作可能變得很困難，所以最好的方法就是預防，也就是永遠使用同樣的名字代表歡迎程式，一般標準的歡迎程式的名字就是**HELLO**。

使用INIT命令

一個典型的**INIT** 命令就如同下面一般：

```
INIT HELLO,S6,D1,V36
```

如您所猜測的，擴充接點，磁碟機及磁碟片號碼都是可以被省略的，如果您將磁碟片號碼包含在命令內，那麼**DOS** 就會把這個號碼放入這個磁片內，**INIT**命令是唯一的一個能夠指定磁碟片號碼的命令，當磁碟片號碼被使用在其他的**DOS** 命令中時，這個號碼必須與**INIT**命令中所指定的相符合；如果您省略了它，那麼**INIT**命令將自動地給這個磁片 254 這個號碼。

像往常一樣，將擴充接點或磁碟機等元素省略掉，會使前面**DOS**指定的值發生作用，因此在您使用**INIT**命令之前，您必須確定那一個磁碟機是被指定的，如果您並不確定，那麼您必須指出它們來。

要設定一個新的磁片的格式，您首先將"**SYSTEM MASTER DISKETTE**"移開磁碟機，將一片新的磁片放入磁碟機內，然後使用**NEW** 命令將記憶體消除乾淨，接著打入上面示範的歡迎程式或您自己設計的歡迎程式，在歡迎程式被存入磁片之前最好先做一個測試，看看這個程式是否運作正常。

現在讓我們指定磁碟片號碼爲 123 ，接著打入：

```
INIT HELLO,S6,D1,V123
```

在您確定磁碟機的門被關好之後，按下**RETURN** 鍵，這時如往常一般地磁碟機的紅燈會亮，而且伴隨著摩擦及轉動的聲音，這整個的過程大約要花兩分鐘的時間，所以您必須稍做忍耐，在燈熄滅之後，您就可以使用**CATALOG** 命令看看在磁片上的東西。而這結果應該與下面的相似

```
DISK VOLUME 123
I 002 HELLO
```

或者您使用的是**APPLESOFT** 語言，您應該看到下面的情形：

```
DISK VOLUME 123
A 002 HELLO
```

I字就表示歡迎程式，是用整數BASIC 語言所設計的，A字則表示使用的語言是APPLESOFT ，002表示佔用了兩個磁區的位置；由於您所設計的歡迎程式長度不同，您可能看到其他的數字。

這時您應該爲您的磁片準備一張標示紙，在上面寫上這個磁片的磁碟片號碼及一些關於磁片的資料；把磁片移開，並將這張紙貼上，如果您想用這張磁片做BOOT 的動作，那麼您隨時都可以開始了。

SAVE命令

如果您一直跟隨著本章所描述的手續執行您的作業，那麼您應該有一片剛被設定格式的磁片放在APPLE Ⅱ內的第6個擴充接點上的第一個磁碟機內，也許在您主機的記憶體內仍然存有您的歡迎程式，您可以使用LIST 命令看看這個程式是否仍然存在，如果已經消失了，您可以打入：

```
LOAD HELLO
```

由磁片上將這個程式重新存入記憶體內。

SAVE 命令將程式存入磁片內；想要將您這個歡迎程式重新複製一份到您的磁片上，您只需打入SAVE 命令，接著打入一個檔的名字就可以了；對於這個例子而言，我們使用GREETING PROGRAM 做爲檔的名字，當然您可以使用任何您喜歡的名字；

```
SAVE GREETING PROGRAM
```

磁碟機就會如往常一般地動作，當SAVE 命令結束時，您將看到BASIC 的系統提示號及游標；接下來，您就可以使用CATALOG

命令檢查磁片的內容了；您所見到的應該與下面的相似：

```
DISK VOLUME 123

 A 002 HELLO
 A 002 GREETING PROGRAM
```

您可以使用這種方法存入任何BASIC 程式。

如果您所使用的名字已經存在磁片上了，那麼不論您所存在的程式是什麼種類，都將取代了原來同樣名字的程式，所以原來的程式也就被消除了；這種情形只會發生在新的及舊的程式都是同一種的BASIC語言型式時，如果它們並不是同一個型式，那麼這個新的程式將不會被存入磁片內，而您將得到**FILE TYPE MISMATCH**的訊息。

DELETE命令

在經過一段時間之後，您也許會在磁片上累積了許多不再被使用程式，**DELETE**命令能夠將程式由磁片上移走。

想要將您前面存入的歡迎程式清除掉，您可以使用下面任何一個命令達到這個目的（除非您在前面指定過其他的磁碟機）。

```
DELETE GREETING PROGRAM,S6,D1,V12
DELETE GREETING PROGRAM,V123
DELETE GREETING PROGRAM
```

記得您可以按照任何順序將擴充接點、磁碟機及磁碟片號碼等元素排列，或者您知道設定的磁碟片正是您所要處理的，那麼您就可以不必寫出任何元素。

LOCK命令

有一些存在磁片上的程式或資料檔必須被永遠地保存起來，爲了達成這個目的，DOS 提供了一個技巧，稱爲鎖住檔（LOCKING FILE），將一個檔鎖住的目的在於防止它被不小心地被消除或是被其他的檔蓋住，您可以輸入LOCK 命令把您的所要保護的檔鎖住，在LOCK 命令之後跟著這個檔的名字及可以省略的磁碟機，擴充接點及磁碟片號碼等元素。下面的這個命令將歡迎程式鎖住：

```
LOCK HELLO
```

將歡迎程式鎖住是一個好主意。

當一個檔被鎖住之後，隨後想要把這個檔消除或是重新寫入資料都會使您得到FILE LOCKED的訊息。

如果被鎖住的檔是一個整數BASIC 或APPLESOFT 的程式，而您想要使用同樣的名字存入不同型式BASIC 語言的另外一個程式，那麼您將得到FILE TYPE MISMATCH的錯誤。

在磁碟的目錄上，被鎖住的檔由一個位於表示檔型式前面的星號（＊）表示。

UNLOCK命令

當您決定要重新寫入或消除一個被鎖住的檔的資料時，您可以使用UNLOCK命令將LOCK 命令消除。下面的這個敍述將歡迎程式看做可以被自由處理的程式：

```
UNLOCK HELLO
```

如往常一樣，擴充接點，磁碟機及磁碟片號碼等元素都是可以被省略的，而且也可以按照任何順序排列。

RENAME命令

您可以更換任何存在磁片上的檔的名字；將一個程式檔存入記憶體內，再將它消除掉，然後再用一個新的名字將它存入磁片內是一種方法，但是更好的方法就是使用**RENAME**命令，**RENAME**命令不論所使用的**BASIC** 型式是那一種，都將發生作用，它對於文字檔及二進位檔也都可以發生作用。下面是一個**RENAME**的例子。

```
RENAME OLDNAME,NEWNAME
```

DOS 將不會檢查這個新的檔的名字是否已經在磁片上了，如果這種情形發生時，您將會得到具有同樣名字的兩個檔，而這一點將使您困惑而且很難恢復的。

除非您將一個新的程式用一個舊的名字代表，否則不要**RENAME**歡迎程式，因爲當這個磁片被**BOOT** 的時候，**RENAME**並不能改變**DOS** 所要尋找的名字。

當一個檔被鎖住時，您也不可以**RENAME**它。

您可以依您的意願指出擴充接點，磁碟機及磁碟片號碼等元素。

VERIFY命令

偶而也許您想要檢查檔的資料是否是完整的，最好的方法就是使用**VERIFY**命令；下面的這個命令將檢查歡迎程式：

```
VERIFY HELLO,V123
```

當**DOS** 將一個檔的資料寫入每一個磁區的時候，**DOS** 將計算一個數字，這個數字就是檢查總數（**CHECKSUM**），檢查總數是基於儲存在磁區內的每個字的數值所計算出來的，而這個檢查總數就被存在

磁區內，當您使用VERIFY命令時，DOS 就會重新計算這個檔中每個磁區的檢查總數，然後與存在磁區內的檢查總數比較，如果這兩個數值有不相同的地方，那麼DOS 就會給您一個I / O ERROR的錯誤訊息。

如果計算出來的值與存在磁區內的值相符合，DOS 不給您任何的訊息，這時您看到的是BASIC 系統提示及游標。

如同其他的DOS 命令一般，您可以在VERIFY命令中使用擴充接點，磁碟機及磁碟片號碼等元素，並且它們可以按照任何順序排列。

在程式中使用DOS命令

直到現在為止所有DOS 的命令都是由鍵盤輸入執行的，但是BASIC程式也可以使用DOS 命令處理檔案。

在BASIC程式中使用DOS 命令，您將它放在一個PRINT 敍述中，並且在前面加上一個ASC Ⅱ碼中 4 的字，而這個字可以使用CTRL-D 造出來，CTRL-D必須是PRINT敍述中第一個被輸出的字，因此您必須確定前面的一個PRINT 敍述不是使用分號或逗號做結束的。

由於CTRL-D所造出的字是不能顯示的，因此我們推荐您在每一次使用CTRL-D的時候，使用一個REM 命令做一個說明，就像下面一般：

```
1000  PRINT "RUN MENU" : REM  THERE IS A CTRL-D
      BETWEEN THE FIRST QUOTE AND THE R IN RUN
```

您可以定義一個變數代表CTRL-D ，然後在PRINT 敍述中使用這個變數，後面再跟著DOS 命令。D $一般被使用來代表CTRL-D ，但是您也可以使用任何您喜歡用的變數；在一個程式中使用標準的變數D$ ，將使得它們具有互相配合的特性。在您的BASIC程式中加入下

面的這一行：

```
10 D$ = "": ?     THERE IS AN INVISIBLE
   CTRL-D BETWEEN THE QUOTES
```

或者，您在**APPLESOFT** 中使用：

```
10 D$ =  CHR$ (4): REM  CHR$(4) = CTRL-D
```

那麼每當**D$** 被使用時，很容易就會發現它是代表$\mathbf{C_{TRL}}$**-D**了。

試著在您的**APPLE Ⅱ**上執行下面這個程式：

```
10 D$ = "": REM .CTRL-D
20  FOR I = 1 TO 10
30  PRINT D$;"CATALOG"
40  NEXT I
50  END
```

除非在您的磁片上存在有18個以上的檔，您將看到十次目錄，而不需要碰觸任何鍵，如果磁片上存有18個以上的檔，那麼在每次顯示暫停的時候，您必須按下任何一個鍵使顯示繼續。

使用磁碟檔

APPLE的**DOS**提供兩種型式的磁碟檔：順序處理檔（**SEQUE-NTIAL FILES**）及隨機處理檔（**RANDOM-ACCESS FILES**），這兩種型式的檔都含有資料塊，被稱爲區（**FIELDS**），一個區可以是一個字或是許多個字組成的。

順序處理檔

順序處理檔就如它的名字一般，它只能依照順序被處理，想要讀或寫檔中的最後一區，您必須首先讀或寫過前面所有的區。對於某些應用而言，順序處理是一個不錯的方法。

隨機處理檔

隨機處理檔較順序處理檔有更大的彈性，您可以不管區的位置而讀或寫它們。對於許多應用而言，隨機處理是最好的方法。

使用順序處理檔

您必須學習下面這些**DOS** 命令才能使用順序處理檔：**OPEN，CLOSE，READ**及**WRITE**。

打開順序處理檔

在磁碟檔被使用之前，必須先被打開（**OPENED**），當一個磁碟檔被打開時，**DOS** 就會去搜尋這個檔的資料，看看它是否存在這個磁片上，如果磁片上有這個檔，那麼這個檔存在什麼位置？**OPEN** 命令同時在記憶體中預留緩衝區的位置。使得您在處理小筆的資料（例如區的資料）時不必趨動磁碟機去尋找，這樣就節省了許多的時間。**OPEN** 命令就如同下面一般：

```
OPEN FILENAME,S6,D2,V99
```

如果這個檔並不在您所指定的磁碟上，**DOS** 將在磁碟索引檔中建

立一個新的檔。

若在程式中使用**OPEN** 命令，則它必須要在**PRINT** 敍述中，並且前面加上**CTRL-D**字。

下面的這個程式只是在磁碟上建立一個名叫**SEQUENTIAL** 的順序處理檔：

```
100 D$ = "": REM  CTRL-D
200  PRINT D$;"OPEN SEQUENTIAL"
300  END
```

我們稱這個程式爲程式1。

你可以在**OPEN** 命令中使用任何擴充接點，磁碟機及磁碟片號碼元素的組合方式：

```
100 D$ = "": REM  CTRL-D
200  PRINT D$;"OPEN SEQUENTIAL,S6,D1,V123"
300  END
```

或

```
200  PRINT D$;"OPEN SEQUENTIAL,V123,S6"
```

需要注意的是所有的元素都必須包含在引號之內，而且它們之間都是使用逗號分開的。

在您執行這個程式之後，**SEQUENTIAL** 這個檔將存在目錄內，您可以使用**CATALOG** 命令確定這件事。磁碟的目錄應該與下面的情形相似：

```
DISK VOLUME 123

*A 002 HELLO
 T 001 SEQUENTIAL
```

注意在**SEQUENTIAL** 檔前面的T字，T字就是代表**SEQUENTIAL** 爲一個文字資料檔（**TEXT FILE**），與**APPLESOFT**或整數**BASIC**的程式檔或二進位檔（**BINARY FILE**）都是不同的，在**HELLO**檔最前面的星號（*）就表示這個檔是被鎖住了的。

關閉檔

事實上，程式 1 是一個極差的程式，因爲它在處理完畢之後沒有將這個檔關閉。**CLOSE** 命令是非常重要的，如果您沒有將打開的檔關閉，往往會使檔內的資料失掉，而且可能毀掉其他磁碟上的資料（參看本章中的軟體毀損一節）。**CLOSE** 命令有兩種格式，第一種是：

```
CLOSE
```

這個**CLOSE** 命令將會把不論在那一個磁碟機內、擴充接點上或具有不同磁碟片號碼的磁碟上的所有打開的檔關閉掉。

有時候您也許希望關閉一個或較多的檔，您可以在**CLOSE** 命令中加上個別檔的名稱關閉這個檔，就如以下一般：

```
CLOSE FILENAME
```

對於這兩種格式的**COSEE**命令都不允許也不需要擴充接點、磁碟機或磁碟片號碼等，只要這個檔是被打開的，**DOS** 就會知道這個檔在什麼地方。

程式 1 應該加上下面這一個程式行：

```
290  PRINT D$;"CLOSE"
```

或者，您可以加入：

```
290  PRINT D$;"CLOSE SEQUENTIAL"
```

寫入順序處理檔

程式 1 是一個沒有什麼用的程式。磁片就是被用來儲存及搜取資料

的，由於磁片上沒有任何資料，您就無法搜取任何東西，所以我們首先討論如何儲存資料。

資料被送入DISK Ⅱ內是與送到螢光幕或列表機上採用相同的方式：使用**PRINT**敍述，任何您可以列印的東西都可以放入磁碟檔中；事實上，您可以視一個順序處理檔如同一個**TV**顯示器，或者視其爲在列表機中的一張紙也許更好一些。

當您輸入一些資料到檔中時，**DOS**修正內部的指標，使它指到磁片上的下一個可供儲存資料的位置，就如同列表機將紙往前移動一行一般。

一個順序處理檔的指標只能往前移動；**OPEN** 命令會把指標移到檔的起始處。

在您輸入資料到一個磁碟檔之前，您首先必須使用**WRITE**命令通知**DOS**，接著的**PRINT**敍述是將資料寫入檔內，而不是顯示在螢光幕上；對於順序處理檔而言，**WRITE**命令就如同下面的一般：

```
WRITE FILENAME
```

在您使用了**WRITE** 命令之後，接著的輸出資料將直接輸出到您指定的檔內；需要注意的是，在輸出資料中包括了錯誤的訊息，但是，當錯誤的訊息被儲存在檔內之後，**WRITE**命令就被取消了，您在螢光幕上只會看到**BASIC**系統標示及游標。

WRITE 命令必須在**PRINT**敍述中使用，前面接著一個**CTRL-D**字，如果您在立即式中使用**WRITE** 命令，您將得到**NOT DIRECT COMMAND** 的錯誤訊息。

在程式1中，加入下面的程式行：

```
210  PRINT D$;"WRITE SEQUENTIAL"
220  PRINT "THIS TEXT WILL BE STORED IN THE FILE"
```

這個程式處理下面的工作：

1. 如果有必要，它建立一個檔。
2. 它將這個檔打開。
3. 它將文字資料存入檔內。
4. 它將這個檔關閉。

您可以依您的需要在行號 210 與 290 之間加入任何數目的**PRINT**敍述，只要**PRINT**敍述的語法是正確的，您可以使用任何組合的文字資料，數字或變數輸入資料。例如：

```
220  FOR I = 1 TO 100
230  PRINT I
240  NEXT I
250  PRINT "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

需要注意的是，您只能在**DOS** 命令之前使用**CTRL-D**，絕不要在輸入資料到檔內時使用。

當**FLASH** 或**INVERSE**敍述在發生作用的時候，不要輸入任何字到檔內，因爲這些字**DOS** 無法適當地處理。

記得當您每一次重新執行這個程式時，不論**PRINT**敍述內存有什麼資料都將會把已經存在檔內的資料覆蓋掉。如果您**PRINT**的字較原先在檔內的字要少，那麼原先資料後面的部份仍然存在，跟在新寫入資料的後面。

想要消除跟在後面不需要的資料方法就是在您存入新資料之前先將舊的資料消除掉；**DELETE**命令可以放在 **OPEN** 命令之前做這個工作，每一次這個程式被執行時，這個檔就被消除掉，然後又被**OPEN**命令建立起來。

但是，如果您想要執行這個程式而**SEQUENTIAL** 這個檔並不存在磁片上，就有一個問題產生了。當這個檔並不存在時，您將得到 **FILE NOT FOUND** 的錯誤訊息，您可以在**DELETE**命令之前加入另外一個**OPEN** 命令做爲預防；下面就是這樣做後發生的情形：

1. 第一個**OPEN** 命令當檔並不存在時，建立這個檔。
2. **DELETE** 命令不論這個檔何時被建立，都會把這個檔消除掉。
3. 第二個**OPEN** 命令將建立一個新的，空的檔。

當您對程式1做了上面的修改之後，您需要記得的是，除非您想要處理其他的磁碟機，否則您前面所指定的擴充接點，磁碟機及磁碟片號碼都不必再在其他的**DOS** 命令中出現的。因此，您先使用一個**OPEN**命令包含了這些元素做為第一個**DOS** 命令，然後，隨後的**DOS** 命令都依從這些指定值。

如果您對程式1做了上面的修改，那麼您的程式應該如同下面一般

```
100 D$ = "": REM  CTRL-D
110  PRINT D$;"OPEN SEQUENTIAL,V123,S6,D1"
120  PRINT D$;"DELETE SEQUENTIAL"
200  PRINT D$;"OPEN SEQUENTIAL"
210  PRINT D$;"WRITE SEQUENTIAL"
220  PRINT "THIS TEXT WILL BE STORED IN THE FILE"
290  PRINT D$;"CLOSE"
300  END
```

當您執行這個程式時，它將把在行號210與290之間的**PRINT**敍述內的資料存入**SEQUENTIAL**檔內；您可以使用任何檔的名字，但是如果您改變了檔的名字，那麼對於每一個涉及到該檔的命令都要把名字改正過來。

您可以使用一個變數代表檔的名字，而在程式內加入詢問程式名字的程式行。做了這樣的修改，這個程式應該和下面的相同。

```
10  INPUT "FILE NAME: ";F$
100 D$ = "": REM  CTRL-D
110  PRINT D$;"OPEN ";F$;",V123,S6,D1"
120  PRINT D$;"DELETE ";F$
200  PRINT D$;"OPEN";F$
210  PRINT D$;"WRITE ";F$
220  PRINT "THIS TEXT WILL BE STORED IN THE FILE"
290  PRINT D$;"CLOSE"
300  END
```

您可以更進一步的詢問擴充接點、磁碟機及磁碟片號碼等元素的資料，而做下面的修正。

```
10   INPUT "FILE NAME: ";F$
20   INPUT "SLOT NUMBER: ";S
30   INPUT "DRIVE NUMBER: ";D
40   INPUT "VOLUME NUMBER: ";V
100 D$ = "": REM  CTRL-D
110  PRINT D$;"OPEN ";F$;",S";S;",D";D;",V";V
```

當您做了這些改變後，您必須依照它所要求的確實輸入，同時在 **S,D,V** 元素之前不要忘了加入逗號，否則**DOS** 將無法分辨出檔的名字及這些元素。

要使這個程式眞的能夠被使用，它應該能讓您輸入想要被儲存的文字資料，而不是每次去改變**PRINT** 敍述，下面是解決的一個方法。

```
150  INPUT "ENTER THE TEXT TO STORE: ";T$
.
.
.
220  PRINT T$
```

但是這個方法只能夠使您輸入並儲存一行的文字資料，您可以加入更多的 **INPUT**敍述，後面跟著**PRINT**敍述做更多的資料輸入，但是輸入及儲存的資料仍然被限制在一定的數目內；爲什麼不在 **INPUT**敍述之後做一個測試的工作，使得在資料結束的時候能夠通知程式呢？因此我們在**PRINT** 之後加入一行**GOTO** 的程式行：

```
150  INPUT "ENTER THE TEXT TO STORE: ";T$
160  IF T$ = "END" THEN 290
210  PRINT D$;"WRITE ";F$
220  PRINT T$
230  GOTO 150
```

現在當您想要結束輸入文字資料的工作時，您只要打入**END** 而這個檔就會被關閉；但是這裏仍然有一個大問題，您應該記得**WRITE**命令使得所有的輸出直接被寫入檔內；由於 **INPUT** 敍述會輸出這樣的訊

息：ENTER THE TEXT TO STORE：，在WRITE命令被執行之後，這一行的訊息將不會顯示在螢光幕上，它將被存在磁碟檔內。

在輸出的資料並不想被寫入磁碟檔內之前，我們必須把WRITE命令取消；任何的DOS 命令都會把WRITE命令取消掉，但是最安全的方法是使用一個虛命令（NULL COMMAND）—那也就是只有C_{TRL}-D這個字。加入下面的程式行：

```
225  PRINT D$
```

在上面所有的改變完成後，現在您就擁有一個可以使您儲存任何數量文字資料的程式了（儲存的極限也就是磁片的容量），您可以使用任何檔的名字，而且使用任何與APPLE Ⅱ連接的磁碟機。

讀順序處理檔

就如同資料可以直接輸出到磁碟一般，輸入的資料可以由磁碟檔中取得。READ 命令指出一個磁碟檔爲輸入資料的來源。對於順序處理檔而言，READ 命令就和下面一樣：

```
READ FILENAME
```

READ 命令必須寫在PRINT 敍述內，而且前面必須加上C_{TRL}-D 字，如果您在立即式情況下使用READ 命令，您將得到NOT DIRECT COMMAND的錯誤訊息。

在READ 命令被執行之後，隨後的INPUT 敍述就會從指定的檔內接受資料，直到另外一個DOS 命令或是一個錯誤取消了READ 命令時爲止。下面的程式2，對於READ 命令的使用做了一個示範：

```
100 D$ = "": REM  CTRL-D
110  INPUT "FILE NAME TO READ: ";F$
120  INPUT "SLOT NUMBER: ";S
130  INPUT "DRIVE NUMBER: ";D
140  INPUT "VOLUME NUMBER: ";V
150  PRINT D$;"OPEN ";F$;",S";S;",D";D;",V";V
160  PRINT D$;"READ ";F$
170  INPUT A$
180  PRINT A$
190  GOTO 170
200  END
```

程式 2 把在程式 1 中所建立的磁碟檔內所有的文字資料顯示出來，在所有的檔內資料被輸出之後，您將看到**END OF DATA** 的訊息，而且這個程式將在**BREAK IN 170** 的訊息時停止。

您也許注意到了程式 2 缺少了一個**CLOSE** 命令，我們曾經解釋過了爲什麼當檔不再被使用時一定要被關閉的原因，但是在上面這個情形時，檔都沒有被寫入資料，所以索引檔或磁軌／磁區內的資料也不需要被修正（假設您指定的檔的名字已經存在磁碟內了）。當然爲了安全起見，這個檔應該要被關閉的。

對於設計一個會產生錯誤訊息及在資料結束時執行會停止但並不在控制下的程式，並不是一個很好的練習。

防止END OF DATA錯誤

在錯誤發生之前，**END OF DATA** 這個情況可以先被偵測出來，程式 2 可以在發現資料結束之後將程式控制轉移到一個**CLOSE** 的命令去；偵測**END OF DATA** 最簡單的方法就是使用**APPLE-SOFT** 中的**ONERR GOTO** 的敍述（在第四章中我們討論過），但是在整數**BASIC**中**ONERR GOTO** 並不能使用（在附錄C中我們將討論**DOS** 的錯誤碼（**ERROR CODES**））。

另外一個發現資料結束的方法就是修改程式 1 ，使得當您在單獨打入**END** 字時，而在檔被關閉之前寫入一個特別的字或碼，然後修改程

式2使得它能夠尋找這個特別的資料結束的標示字，然後在尋找到這個字時正常地**CLOSE**掉這個檔。這個方法可以在整數BASIC或APPL-ESOFT中同樣發生作用。

整數BASIC與Applesoft的區別

READ 命令指定了一個磁碟檔做為接著的**INPUT**敍述的資料來，但是**INPUT**敍述的語法必須遵從所使用的**BASIC**語言規則；也就是說，由檔內所得到的資料完全依照它被儲存時的格式。

考慮下面這個程式：

```
100 D$ = "": REM  CTRL-D
200  PRINT D$;"OPEN FILE1"
300  PRINT D$;"WRITE FILE1"
400  PRINT "HELLO"," THIS IS A TEST"
500  PRINT D$;"CLOSE"
600  END
```

在**APPLESOFT**中執行這個程式，然後改變行號300及400以便讀取在**FILE1**中的文字資料：

```
300  PRINT D$;"READ FILE1"
400  INPUT A$
```

這時**A$**內放的是**HELLO THIS IS A TEST**。現在將原先的程式改寫為：

```
300  PRINT D$;"WRITE FILE1"
400  PRINT "HELLO, THIS IS A TEST"
```

再執行它；現在如前面一般地讀取這個文字資料，這一次您將得到?**EXTRA IGNORED** 的錯誤訊息，而**A$**內放的是**HELLO**。

在**APPLESOFT**中，逗號把在一個**PRINT**敍述中的許多個值分開。

至於整數**BASIC** 中則認為這兩行都是可以被接受的，所以在第

一個例子中A $ 的值就是HELLO THIS IS A TEST ，而在第二個例子中的A $ 則是HELLO ,THIS IS A TEST 。

在Applesoft中使用GET敍述讀取文字資料檔

在APPLESOFT中您可以使用GET 敍述自磁碟檔內讀取資料，B GET 敍述與INPUT敍述的差別在於GET 敍述每一次只讀取一個字，因此，如果一個檔存有THIS FILE CONTAINS 的文字資料，而您執行一個OPEN，一個READ，然後執行GET 命令，那麼第一個GET 敍述將只會讀回T字，第二個GET 則讀回H字，而隨後的GET 敍述將會把 I ,S,空白，F, I,L,E 等字讀回，直到所有存在檔內的字被讀完爲止。如果GET 敍述讀回了一個逗號或是一個游標回頭，那麼這個程式就可以偵測出來而且對它做適當的處理。

下面顯示的程式 3 ，對於如何使用APPLESOFT 中的GET 敍述讀取一個檔而建立一行的文字資料－每次一個字，做了一個說明。

```
100 D$ =  CHR$ (4): REM  CTRL-D
200  INPUT "FILE NAME TO READ: ";F$
300  INPUT "SLOT NUMBER: ";S
400  INPUT "DRIVE NUMBER: ";D
500  INPUT "VOLUME NUMBER: ";V
600  PRINT D$;"OPEN ";F$;",V";V;",D";D;",S";S
700  PRINT D$;"READ ";F$
800 B$ = ""
900  GET A$
1000  IF A$ =  CHR$ (13) THEN 1300
1100 B$ = B$ + A$
1200  GOTO 900
1300  REM  RETURN CHARACTER FOUND
1400  REM  B$ IS COMPLETE
1500  PRINT B$
1600  GOTO 800
1700  END
```

但是GET 敍述被使用在磁碟檔時有一個問題，那就是在GET 敍述被執行之後第一個要被印出的字會被取消，如果第一個要被印出的字是一個DOS 命令，那麼C_{TRL}-D 這個字將不會被承認，那也就是說

這整個的命令將被印出而不會被當作**DOS** 命令來執行。

對於解決這個問題的方法就是首先印出一個可以被丟棄的字，也就是故意要被忽視的字。**C_{TRL}-A**（**ASC**Ⅱ碼的1）就是這麼一個字，它無法被印出也沒有什麼特別的意義。

程式3可以將行號1500 處修正後得到正確的程式：

```
1500  PRINT  CHR$ (1);B$: REM  CHR$(1) = CTRL-A
```

將數字存在檔內

您也許已經使用過了程式，把數字儲存到一個檔內，它也許是文字資料的一部份，如下面一般：

```
220  PRINT "MY ADDRESS IS 1234 NORTH STREET"
```

或者直接就用數字值：

```
230 PRINT 1,2,3,4,5
```

或者經過數值變數

```
240  PRINT A,B,C,D,E
```

如果您直接儲存數字，當您使用程式2將它們讀回時將得到奇怪的結果。

在整數**BASIC** 中，數字用逗號或分號分開會形成一個大的數字（行號230 的12345 ），而做為一行的輸出。

在**APPLESOFT** 中，事情會變得更令人迷惑了，前面三個數字會被連接在一起而做爲一行的輸出，至於後面兩個值（ 4及5 ）則各自佔用一行的位置輸出，因此共有三行的輸出。

這些問題都是由於存資料到磁碟檔內的格式所造成的；如果逗號沒有被顯示在螢光幕上，那麼它們就不會被放在數字之間，至於螢光幕上

的逗號，就會使得下一個值出現在下一個TAB 的位置；但是當直接輸出到磁碟檔時，DOS 將把逗號完全地丟棄，預定位置的情形將永遠不會發生；而結果是，所有的值都將連接在一起直到一個游標回頭（ASCⅡ碼13）將它們分開爲止，DOS 只認爲游標回頭是一個可以分開值的字。在整數BASIC中在第五個預定位置產生後（逗號），會產生一個游標回頭，至於APPLESOFT 則在第三個預定位置後就產生游標回頭了。

爲了避免問題發生，您應該確定在每一個存入磁碟檔內的數值都接著一個游標回頭字。最簡單的方法就是使用分開的PRINT 敍述輸出各別的數字：

```
230  PRINT 1: PRINT 2: PRINT 3: PRINT 4:
 PRINT 5
```

另外一種可以在APPLESOFT 中使用的方法就是將游標回頭字加入PRINT敍述內做爲一個分界字：

```
230  PRINT 1;CHR$(13);2;CHR$(13);3;CHR$(1
 3);4;CHR$(13);5
```

您也可以指定一個變數做爲游標回頭字，然後印出這個變數：

```
11  R$ = CHR$(13): REM  RETURN CHARACTER
    .
    .
    .
230  PRINT 1;R$;2;R$;3;R$;4;R$;5
```

這使得您的程式看起來較爲簡捷。

如何在順序處理檔內增加資料

一個順序處理檔在資料寫入之後，必須被關閉，當您CLOSE

個檔之後，您就不知道最後一個項目存在何處了；因此，您想要在檔的後面加入資料，首先必須找到這個檔的結尾；您可以按順序地讀每一項資料直到最後一個為止，但這對於具有大筆資料的檔而言是非常浪費時間的。APPEND命令所做的就是這項工作。

APPEND命令將檔的指標放在檔的結尾後的第一個字的位置；如果您在APPEND命令被執行後讀取資料，您將得到END OF DATA的錯誤，如果您在APPEND命令後寫入資料，那麼這個新的資料將被加在這個檔的後面。

APPEND命令可以用來代替OPEN 命令，在APPEND及OPEN 命令間有兩個重要的差別處：

1. APPEND 需要已經存在的檔，如果這個檔並不存在，那麼您將得到FILE NOT FOUND 的錯誤訊息，APPEND不會建立新的檔，它只假設檔是已經存在的。
2. APPEND 將指標放在檔的結束處，OPEN 則將指標放在檔的開始處。

APPEND 命令的格式與OPEN 命令的相同，下面就是一個例子：

```
APPEND FILENAME,S6,D2,V99
```

如往常一般，擴充接點、磁碟機及磁碟片號碼等元素都是可以被省略的。

POSITION命令

另外一個有用的命令就是POSITION 命令，POSITION 將指標往前移動（絕不往後），依照現在指標的位置加上指出數字的區的數目往前移動。POSITION 看起來就像下面一般：

```
POSITION FILENAME,R30
```

R指出相關的區，而在R後面的數字就代表所要跳過的區的數目，區是由游標回頭字做爲分界的，所以上面的POSITION命令計算30次的游標回頭然後將指標移到那個位置，如果您使用的是R 0，那麼指標就不會被移動。

POSITION 確實地按照一個字一個字地由目前的位置開始檢查，如果區的數目不夠，或是遇到一個沒有被使用到的數元組，那麼您將立即得到END OF DATA的錯誤訊息，執行INPUT或GET 敍述則不會產生這種錯誤。

在使用POSITION 命令之前，檔必須被打開；當您打開一個檔之後，指標就會指著檔的起始處，如果您在指標指著檔的起始處時使用POSITION命令，它就等於選擇這個檔的絕對位置的區。

記得的是，POSITION與其他的DOS 命令一般，它將取消READ及WRITE 命令，所以您在使用POSITION命令的時候，必須確定它是被使用在READ或WRITE 命令之前，而不是在後面。

使用隨機處理檔

隨機處理檔是將資料分割成一個一個的記錄（RECORD）而組成的；在一個檔內，每一個記錄存有相同數量的資料，當這個檔被建立時，這個數量就被改訂爲若干數目的數元組了。

能夠被儲存在一個記錄內的資料長度就被稱爲記錄長度。

記錄使用一個數字代表它在檔內絕對的位置，在每一個檔內的第一個記錄就用數字Ø來代表它，接著就是數字1，然後依序下去。

最小的隨機處理檔具有一個記錄，檔在新的記錄被加入時擴增，但並不縮小，想要移開不必要的記錄而縮小檔的長度，您必須將想要保存下來的記錄複製到另外一個新的隨機處理檔內。

程式中在使用這個檔時，必須指出要使用的記錄是那一個，而且要被處理的是這個記錄的那一部份。

打開隨機處理檔

定義一個隨機處理檔，您必須在檔被打開時，加入一個元素：長度元素；長度元素（**L**）指出這個記錄的長度，就如下面一般：

```
OPEN FILENAME,L10,S6,D1,V100
```

長度元素的值必須在 1 與 32767 之間，它不必是這個命令的第一個元素，但如果這個檔是隨機處理檔的話，它必須要寫在這個命令中。

在程式中絕不可以將超過長度元素值的資料寫入記錄內，這個長度包含了游標回頭及逗號。如果太多的字被存在記錄內，那麼隨後的記錄可能被覆蓋或者與其他的記錄連接在一起，那就造成了極大的混亂。

將隨機處理檔關閉

CLOSE命令在順序處理檔及隨機處理檔內是相同的。

隨機處理的READ及WRITE

READ及**WRITE** 命令在處理隨機處理檔時需要一個記錄元素。這個記錄元素將檔的指標移到某個記錄的起始處。下面這個例子使用了記錄元素（**R**）。

```
    READ FILENAME,R13
或
    WRITE FILENAME,R6
```

這個記錄元素不必是這個命令中唯一的元素，您也可以指出擴充接

點、磁碟機及磁碟片號碼等元素，這些元素可以依照任何順序排列；如果記錄元素並沒有出現在命令中，那麼這個指標將不會移動。

一個實際的隨機處理的例子

接下去的程式對於實際上使用隨機處理檔做了一個示範，這些程式在APPLESOFT 及整數BASIC中同樣有效，但是在整數BASIC中，您必須在行號100之前加入變數B$ 及C$(都是255個字的長度）的DIM敘述。第一個程式建立了一個叫做RANDOM的檔。

```
 10  REM       RANDOM-ACCESS FILE CREATING PROGRAM
 20  REM
 30  REM       BE SURE TO RUN THIS PROGRAM FIRST AS IT
 40  REM  STORES INFORMATION IN RECORD ZERO WHICH MUST
 50  REM  EXIST FOR THE NEXT PROGRAM TO FUNCTION.
 60  REM
100  D$="" : REM  CTRL-D
200  PRINT D$;"OPEN RANDOM,S6,D1,L256" : REM  OPEN THE FILE
300  PRINT D$;"WRITE RANDOM,R0" : REM  WRITE, RECORD ZERO
400  PRINT 0 : REM  STORE A ZERO IN RECORD ZERO
500  PRINT D$;"CLOSE" : REM  CLOSE THE FILE
600  END
```

當這個檔被建立之後，第二個程式允許您去讀、修改、加入及列入這個檔中的記錄，每一個記錄都將存有一行由您自鍵盤輸入的訊息。

```
 10  REM       RANDOM-ACCESS FILE DEMONSTRATION PROGRAM
 20  REM
 30  REM       THIS PROGRAM WILL MAINTAIN A RANDOM-ACCESS
 40  REM  FILE WHICH CONSISTS OF SINGLE LINE RECORDS.
 50  REM
 60  REM       RECORD ZERO CONTAINS A NUMBER INDICATING
 70  REM  THE LAST RECORD NUMBER IN USE.
 80  REM
 85  REM  NOTE:  FILE "RANDOM" MUST BE CREATED BEFORE
 90  REM         ATTEMPTING TO USE THIS PROGRAM.
 95  REM
100  D$="" : REM  CTRL-D
200  PRINT D$;"OPEN RANDOM,S6,D1,L256"
225  PRINT D$;"READ RANDOM,R0" : REM  READ RECORD ZERO
250  INPUT M : REM  M = THE LAST RECORD NUMBER IN USE
```

```
275   PRINT D$ : REM  CANCEL READ COMMAND
300   CALL -936 : REM  CLEAR SCREEN
400   PRINT "RANDOM-ACCESS FILE DEMONSTRATION"
500   PRINT : PRINT : PRINT : PRINT
600   PRINT "COMMANDS:" : PRINT
700   PRINT " 0  =  STOP"
800   PRINT " 1  =  READ A RECORD"
900   PRINT " 2  =  ADD A RECORD"
1000  PRINT " 3  =  CHANGE A RECORD"
1100  PRINT " 4  =  LIST ALL RECORDS"
1200  PRINT : PRINT
1300  PRINT "WHICH";
1400  INPUT C
1500  IF C=0 THEN 8300 : REM   BRANCH
1600  IF C=1 THEN 2200 : REM   TO
1700  IF C=2 THEN 3300 : REM   THE
1800  IF C=3 THEN 4600 : REM   SELECTED
1900  IF C=4 THEN 6700 : REM   ROUTINE
2000  GOTO 300         : REM   (OR RE-DISPLAY THE MENU)
2050  REM
2100  REM  * * * * *      READ A RECORD     * * * * *
2150  REM
2200  CALL -936 : REM  CLEAR SCREEN
2300  PRINT : PRINT "READ A RECORD" : PRINT
2400  PRINT "WHICH RECORD NUMBER (0 TO STOP)";
2500  INPUT R
2600  IF R<1 THEN 300 : REM  RETURN TO MAIN MENU
2650  IF R>M THEN 2200 : REM  RECORD DOES NOT EXIST
2700  PRINT D$;"READ RANDOM,R";R : REM PREPARE TO READ RECORD
2800  INPUT B$ : REM  READ THE DATA
2900  PRINT D$ : REM  CANCEL READ COMMAND
3000  PRINT : PRINT B$ : PRINT : REM  DISPLAY THE DATA
3100  GOTO 2400 : REM  ASK FOR ANOTHER RECORD NUMBER
3150  REM
3200  REM  * * * * *      ADD A RECORD      * * * * *
3250  REM
3300  CALL -936 : REM  CLEAR SCREEN
3400  PRINT : PRINT "ADD A RECORD" : PRINT
3500  PRINT "NEXT RECORD NUMBER = ";M+1
3600  PRINT : PRINT "ENTER DATA FOR RECORD ";M+1
3625  PRINT "(PRESS [RETURN] NOW TO STOP ADDING)"
3700  INPUT B$ : REM  GET USER'S RESPONSE
3750  IF B$ = "" THEN 300 : REM  QUIT IS JUST [RETURN]
3800  PRINT D$;"WRITE RANDOM,R";M+1 : REM  PREPARE TO WRITE
3900  PRINT B$ : REM  SEND DATA TO FILE
4000  M = M + 1 : REM  INCREMENT LAST RECORD NUMBER
4100  PRINT D$;"WRITE RANDOM,R0" : REM  PREPARE TO WRITE
  RECORD ZERO
4200  PRINT M : REM  STORE UPDATED VALUE
4300  PRINT D$ : REM  CANCEL WRITE COMMAND
4400  GOTO 3500 : REM  LOOP FOR ANOTHER RECORD
4450  REM
4500  REM  * * * * *      CHANGE A RECORD     * * * * *
4550  REM
4600  CALL -936 : REM  CLEAR SCREEN
4700  PRINT : PRINT "CHANGE A RECORD" : PRINT
```

```
4800  PRINT "CHANGE WHICH RECORD (0 TO STOP)";
4900  INPUT R
5000  IF R<1 THEN 300 : REM  RETURN TO MAIN MENU
5050  IF R>M THEN 4600 : REM  TRY AGAIN IF RECORD NOT ON FILE
5100  PRINT D$;"READ RANDOM,R";R : REM  PREPARE TO READ
5200  INPUT B$ : REM  READ THE RECORD
5300  PRINT D$ : REM  CANCEL READ COMMAND
5400  PRINT : PRINT B$ : PRINT : REM  DISPLAY THE DATA
5500  PRINT "ENTER THE NEW DATA"
5600  PRINT "(PRESS [RETURN] NOW TO CANCEL CHANGES)"
5700  PRINT
5800  INPUT C$ : REM  GET USER'S RESPONSE
5900  IF C$>"" THEN 6200 : REM  BRANCH IF NEW DATA
6000  PRINT "RECORD ";R;" UNCHANGED ! ! !" : REM  LOOP IF
6100  GOTO 4800 : REM  NO CHANGES DESIRED
6200  PRINT D$;"WRITE RANDOM,R";R : REM  PREPARE TO WRITE
6300  PRINT C$ : REM  STORE CHANGED DATA
6400  PRINT D$ : REM  CANCEL WRITE COMMAND
6500  GOTO 4800 : REM  LOOP FOR ANOTHER RECORD TO CHANGE
6550  REM
6600  REM  * * * * *      LIST ALL RECORDS      * * * * *
6650  REM
6700  CALL -936 : REM  CLEAR SCREEN
6800  PRINT : PRINT "LIST ALL RECORDS" : PRINT
6900  R = 0 : REM  RESET THE COUNTER
7000  R = R + 1 : REM  INCREMENT THE COUNTER
7100  IF R > M THEN 7700 : REM  STOP AFTER LAST RECORD
7200  PRINT D$;"READ RANDOM,R";R : REM  PREPARE TO READ
7300  INPUT B$ : REM  READ THE DATA
7400  PRINT "RECORD NUMBER = ";R : REM  DISPLAY RECORD NUMBER
7500  PRINT B$ : PRINT : REM  DISPLAY RECORD'S DATA
7600  GOTO 7000 : REM  LOOP FOR NEXT RECORD
7700  PRINT D$ : REM  CANCEL READ COMMAND
7800  PRINT : PRINT "* * * * END-OF-FILE" : REM  PRINT
 EOF MESSAGE
7900  PRINT "PRESS RETURN TO CONTINUE"; : REM  REQUEST
 RESPONSE
8000  INPUT B$ : REM  GET USER'S RESPONSE
8100  GOTO 300 : REM  RETURN TO MAIN MENU
8150  REM
8200  REM  * * * * *      STOP PROGRAM      * * * * *
8250  REM
8300  PRINT D$;"CLOSE" : REM  CLOSE THE FILE
8400  CALL -936 : REM  CLEAR SCREEN
8500  FOR I=1 TO 24 : PRINT : NEXT I : REM  MOVE TO BOTTOM
 LINE
8600  PRINT "PROGRAM COMPLETE."
8700  END
```

BYTE(數元組)元素

Byte 是隨機處理檔中另外一個有用的元素，Byte （即數元組）元素與**READ，WRITE，POSITION**命令合用，可以將指標移到某個記錄中被指定的數元組去，逗號及**B**字可以和**READ，WRITE**，或**POSITION**命令合用，而記錄元素必須與數元組元素同時出現。例如：

```
READ FILENAME,R19,B3
```

在這個例子中，讀資料的動作會在第20個記錄的第 4 個數元組處開始（記得，第一個數元組是ø）數元組元素可以在記錄中把指標的位置向前或向後移動。

如果您想要使用數元組元素，則記錄必須要有一個確實的資料格式，您必須知道存在記錄內數元組的位置，不然您可能得到的是毫無意義的資料。

當在**POSITION**命令中使用數元組元素時，您必須記得**POSIT-ION** 命令會將前面的**READ**及**WRITE** 命令取消；您可以執行一個含有記錄元素的**READ**或**WRITE** 命令，然後使用**POSITION**命令將指標移到正確的區內，但是您必須接著執行另外一個**READ**或**WRITE** 命令（由於**POSITION** 命令將前面的**READ**或**WRITE**命令取消）。

POSITION只能夠在記錄內將指標往前移動到某一個區內，而數元組元素却可以使您一次一個數元組地去處理這個區內的資料。

其他的DOS命令

下面還有一些尚未解釋 的DOS 命令，它們分別是EXEC ,MAX-FILES , TRACE , MON及NOMON。

EXEC命令

EXEC 是一個非常特殊的命令，EXEC 使您能夠將控制權由APPLE Ⅱ 轉移到一個文字資料檔內。EXEC 如下：

```
EXEC FILENAME,R6,S5,D2,V23
```

R元素與POSITION 命令中的記錄元素相似，它指出EXEC檔內的相關區域，而由這個區域開始處理，由於EXEC 命令將EXEC檔打開，並且把指標放在這個檔的開始處，所以R元素指到的就是這個檔內的絕對位置的區的數目。R∅是一個設定值，設定執行是由這個檔的起始處開始的，R 1 則表示執行由第二個區開始。

至於相關的記錄，擴充接點，磁碟機及磁碟片號碼等元素都是可省略的，而且可以依照任何順序排列。

當 EXEC 命令被執行時，指定的這個檔就被打開了，然後隱藏在這個檔內的READ 及 INPUT敍述就發生作用，這時如果您沒有使用R元素，那麼這個檔的第一個程式行就被取出，如果您使用了R元素，那麼就取用被R所指定的行。

這個被取用的程式行被當做在立即式情況下由鍵盤輸入的命令一般，如果這一行是一些無用的垃圾，那麼您將得到？SYNTAX ERROR或***SYNTAX ERR 的錯誤訊息，如果它是一個有效的程式行：

```
100 PRINT "THIS IS A TEST"
```

那麼就會被存在記憶體中，如同您剛剛打入一般；如果它是一個立即式的命令（**DIRECT COMMAND**），如**LIST**或**RUN**，那麼它將被立刻執行。

在第一個程式行的動作結束之後，隨著就是下一個程式行被讀取然後被執行，這樣子繼續下去直到這個檔結束爲止，在這個時候，**APPLE Ⅱ**的控制權就又轉回到鍵盤去了。

假如一個叫做**BUBBA** 的文字資料檔擁有下面的這些文字資料行：

```
PRINT "I HAVE CONTROL OF YOUR APPLE ! ! !"
FP
100  GR
200  FOR I=0 TO 39
300  FOR J=0 TO 39
400  COLOR=RND(0)*15
500  PLOT I,J
600  NEXT J
700  NEXT I
800  FOR I=1 TO 5000 : NEXT I
900  TEXT : CALL -936
999  END
LIST
FOR I=1 TO 5000 : NEXT I
RUN
NEW
PRINT "FINISHED.  HERE'S YOUR DISK CATALOG:"
CATALOG
```

那麼當您打入下面這個命令後：

```
EXEC BUBBA
```

就會有幾件事情發生了，首先是一個訊息顯示出來，接著**APPPLESOFT** 語言系統就進來了，然後就是一個簡短的程式被存入記憶體內並且顯示出來，這時在一陣短暫的暫停之後，這個程式就被執行，執行完畢後從記憶體中被消除，這時另外一個訊息顯示在螢光幕上，下面跟著目前正在使用的磁碟內的目錄。

EXEC需要注意的事項

EXEC 有幾個有趣的概念，您必須知道。

只有一個EXEC 檔可以在任何時間內被打開，如果一個含有EXEC 命令的檔，被EXEC 命令處理，那麼第一個檔就會被關閉而第二個檔就取得了控制權，在EXEC 檔執行了一個程式後，下一個存有EXEC 的程式行就被執行。如果這個程式執行時，您使用C_{TRL}-D將它放棄，那麼存在其中的EXEC 也不會繼續了。

如果一個程式是由EXEC 命令來執行的，而這個程式含有INPUT 敍述，那麼這個輸入的資料會由EXEC 檔的下一個區內取得，這就產生了一個問題，如果這個被取用的區內放的是立即式的DOS 命令（不是一個程式行），那麼這個命令被執行而不是當做IPPUT 資料被接受。

如果一個程式執行CLOSE 命令，或者EXEC 檔內存有一個立即式的CLOSE 命令，EXEC 檔將不會被關閉。

使用EXEC將一個程式由一種BASIC轉換成另一種BASIC語言

EXEC 可以用來做爲整數BASIC與APPLESOFT 間的轉換工具；如果您想要由APPLESOFT 轉換成整數BASIC ，您首先必須將APPLESOFT 程式編輯成合乎整數BASIC 語法的程式，然後將這個程式當做一個文字資料檔存起來；您可以加入幾個程式行在這個程式內建立文字資料檔，執行一個WRITE命令，然後顯示出這個程式，這個程式的顯示就會進入文字資料檔內了。下面就是這些必須的敍述：

```
 1 POKE 33,33: REM DISABLE "LIST" FORMATTING
 2 D$ = "": PRINT D$"OPEN FILE": PRINT D$;"WRITE FILE"
 3 LIST 10,32767
 4 PRINT D$;"CLOSE": END
 5 POKE 33,40: REM RESET "LIST" FORMATTING
10 REM YOUR PROGRAM BEGINS HERE
```

當您執行這個程式時，只有1,2,3,4,5 行號的敍述被執行，它們將會打開一個檔，然後將這個程式的程式列存入磁碟內，當這些做完後，您就可以改變到另外一種BASIC中了（例如，使用FP或INT，然後打入：

```
EXEC FILE
```

這時這個程式就會從磁碟抄入記憶體內，如果您仍沒有做好修正的工作，您現在必須從事編修的工作改正程式以便適合新語言的語法。

MAXFILES命令

MAXFILES 允許您指出能夠一次被打開的檔的最大數目。每一個檔在記憶體中佔用了595個數元組做爲緩衝區，每一個緩衝區有兩個256 Byte 的段（SECTION），一個用來讀，另外一個用來寫，剩下的83個數元組被用來儲存一些檔的資料，例如磁軌／磁區表。

當一個檔被打開了，或者一個像CATALOG或LOCK的磁碟動作發生了，DOS 就從磁碟內讀取256個數元組的訊息放入緩衝區內，如果這個訊息被改變了而且需要被寫回磁片上去，這時就會從緩衝區內複製256 個數元組的資料並把它存回磁片上去。

下面這個例子指出最多可以一次打開八個檔：

```
MAXFILES 8
```

檔的數目必須是在1與16之間的整數；當DOS 被存入記憶體時，自動就預留了三個緩衝區；如果您同時想使用超過三個的檔，您就可以使用MAXFILES 將能使用的檔的數目定得多一些，如果您想多使用一些記憶體，那麼您也可以使用MAXFILES 將緩衝區數目定得少一些。

在執行下面這些DOS 命令時，只有一個緩衝區會被使用：

APPEND	FP	POSITION
BLOAD	INIT	REAC
BRUN	INT	RENAME
BSAVE	LOAD	RUN
CATALOG	LOCK	SAVE
CHAIN	MAXFILES	UNLOCK
CLOSE	MON	VERIFY
DELETE	NOMON	WRITE
EXEC	OPEN	

如果被您打開的檔的數目達到了極限，然後您使用CATALOG命令，這時您將得到NO BUFFERS AVAILABLE的錯誤訊息；但是，如果不是DOS 的命令，就不需要緩衝區（例如，磁帶的LOAD）。

當MAXFILES 的數目被改變時，整數BASIC程式就會被消除，而APPLESOFT中的字串就會被改變，因此，一定要在執行程式或將程式存入記憶體之前使用MAXFILES 命令（如果您要使用它的話）

MAXFILES 命令可以在APPLESOFT 程式中執行，不過必須放在PRINT敍述內，而且前面接著C_{TRL}-D字。MAXFILES將使得GOTO ，GOSUB 及其他的指令產生不正常的功能，除非它是程式中的第一個敍述才可以避免這種情形。為了避免不正常的情況，您必須照下面的方式使用MAXFILES ：

```
]1 REM FIRST ISSUE MAXFILES COMMAND
]2 PRINT CHR$(4);"MAXFILES 9"
]3 REM REGULAR PROGRAM BEGINS HERE
]4 D$=CHR$(4):REM CTRL-D CHARACTER
```

在整數BASIC中您只能在EXEC檔中使用MAXFILES ；例如EXEC 檔內包含了三行立即式的敍述：

```
MAXFILES 8
LOAD PROGRAM
RUN 10
```

它在下面這個程式中被使用．

```
5  PRINT D$;"EXEC MXF": REM SET MAXFILES
   CONTINUE AT LINE 10
10 REM PROGRAM BEGINS HERE
```

使用DOS的偵錯工具

由於處理磁碟檔的程式往往都蠻複雜的，DOS 有三個命令幫助您找出程式的錯誤：MON，NOMON及TRACE 。

MON命令

MON是MONITOR的縮寫，MON允許您監督由磁碟進出的訊息，MON 使用三個元素。下面的例子使用了這三個元素：

```
MON C,I,O
```

MON 的元素指出所要顯示的訊息的型式；C 使得使用磁碟的命令能夠被您監督，I 使得由磁碟輸入的資料被顯示出來，而O 則使輸出到磁碟的資料被顯示在螢光幕上。

這些元素可以依照任何順序排列或任何組合方式使用，不過最少要有一個元素出現在命令中，否則這個命令將不被執行。

MON 將會一直產生作用，直到NOMON，INT或FP 命令被執行或者DOS 被重新存入記憶體內為止。

NOMON命令

NOMON命令取消MON 命令的功用，它與MON命令一般使用同樣的三種元素，只是NOMON 元素指出那一種資料是不需要被監督的；

例如，假設MON I,O,C 被執行了，這個命令：

```
NOMON O
```

將會把輸出到磁碟的監督工作取消，至於由磁碟輸入及DOS 命令的監督情況仍然繼續顯示出來。

使用TRACE命令

TRACE 命令（參看第四章）被用來偵測程式的錯誤，但是由於TRACE不使用游標回頭而印出行號，因此DOS 命令會與行號連接在一起而不被處理，對於這種情形的補救方法是非常簡單的，也就是在DOS 命令之前先輸出游標回頭。改變下面這一行來做這個工作：

```
100 D$ = CHR$ (4): REM CTRL-D
```

變成：

```
100 D$ = CHR$ (13) + CHR$(4): REM RETURN + CTRL
```

現在只要PRINT D$發生時，CTRL-D前面就接著一個游標回頭，將CTRL-D與行號分開。

另外一個仍然存在的問題，那就是如果您使用了一個WRITE命令，那麼所有的行號都將被存在這個檔內（抱歉的是，對於這個問題尚沒有解決的方法）。

機器語言的磁碟檔

DISK Ⅱ內也可以儲存機器語言及二進位檔（BINARY IMAGE FILES）也就是圖形檔（GRAPHICS FILES），這些檔是用B

字來代表它們的檔的型式。

不論低解析度（LOW-RESOLUTION）及高解析度（HIGH-RESOLUTION）的圖形都可以存在磁片內供以後使用。

機器語言的程式可以被直接地存入記憶體內並且被執行，或者也可以在BASIC程式中使用CALL 敍述或USR功能呼叫它。

DOS 有三個專門爲二進位檔所設計的命令，它們是BSAVE，BLOAD及BRUN，它們的功能與一般檔所使用的相同（BSAVE = SAVE，BLOAD =LOAD，BRUN=RUN）。

BSAVE命令

BSAVE命令，如同它的名字一般，就是把一個二進位的區域存入磁碟內，例如：

```
BSAVE FILENAME,A378,L21,S6,D2,V6
```

需要注意的是，這個命令有兩個元素在其他的DOS 命令中是不存在的，而且這兩個元素不能被省略，它們必須被指出；至於擴充接點、磁碟機及磁碟片號碼等元素都如往常一般可以被省略。

元素A就是位置元素，它指出這個二進位區域存在記憶體中的開始位置，這個位置可以是十進位的或十六進位的常數；十六進位的數值前面必須加上$符號，十進位數值則必須在ø與65536之間，至於負數是不允許的。

L元素則指出這個二進位區域的長度，也就是這個區域所佔用的數元組數，它也可以使用十六進位或十進位數表示，只是十六進位數值之前必須加上$符號，至於它的數值必須在1與32767之間，否則您將得到SYNTAX ERROR的錯誤訊息。32767 個數元組是DOS 所能儲存一個區的最大極限，如果想要儲存超過32767 的一個區，那麼必須使用兩個BSAVE 敍述。

如果您所指出的範圍內有些實際上並不存在的記憶體，這並不會產生任何錯誤，但是指出一個超出您的APPLE Ⅱ所擁有記憶體範圍的位置並沒有什麼好處（例如，在一個48K的系統中指出 49151 或$ BFFF的位置）。

BLOAD命令

BLOAD 由磁碟內取出二進位檔並把它放入記憶體內，BLOAD命令就如下面一般：

```
BLOAD FILENAME,A378,S6,D2,V6
```

BLOAD命令中位置元素可以被省略，如果這個元素被省略，那麼這個二進位檔將被存放在與存入磁碟內時的同樣的記憶體位置中。

如果機器語言的程式沒有被存入適當的位置，那麼它可能不會正常作業。

BLOAD不像LOAD 一般會將記憶體中的程式及資料值消除，除非剛好佔用了您將要存放的位置，只有這些佔用了位置的值或程式才會受到影響，至於其他的記憶體內的資料都將不會改變。

如果您指出的位置是ROM 的位置，也不會有任何的錯誤訊息產生，只是ROM 位置內的值並不會因而受到改變罷了！

BRUN命令

BRUN與BLOAD 幾乎是相同的，只是在檔被存入記憶體內後，BRUN執行一個機器語言的JMP命令，跳到檔的起始位置；如果您沒有指定任何位置，那麼 JMP 將跳到這個區域的被保留起來時的起始位置。下面就是BRUN 的例子：

```
BRUN FILENAME,A378,S6,D2,V6
```

絕對不要使用BRUN 處理圖形映像，因爲它們的結果是不可預測的。

6



圖形與聲音

APPLE Ⅱ具有顯示彩色圖形及發聲的能力，將這些功能加入您的程式之中，就如同它們都是由您設計出來的一般。這一章適合於BASIC 語言的初學者（也許是讀了本書才開始的）也適合於編譯語言的程式設計師。圖形的控制並不困難，尤其是您使用高階層的程式語言時，APPLE Ⅱ的監督程式利用它的一些固定存在的機器語言副程式幫助編譯語言程式設計師使用圖形及發聲器的功能。只要您讀完了本章之後，您就會有足夠的能力使用這些功能了。

低解析度圖形

APPLE Ⅱ在記憶體中爲低解析度圖形（LOW-RESOLUTION GRAPHICS ）預留兩塊位置，它們就被稱爲頁（ PAGES ），每一個低解析度的頁能夠在螢光幕上顯示 40 行乘 48 列的圖形，就如圖 6 － 1 的一般。

行與列的每一個交點在螢幕上就是一個小的長方形，每一頁具有

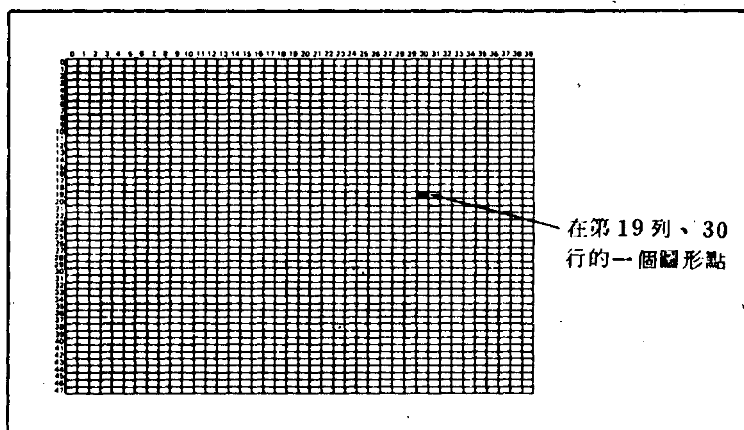


圖 6－1 低解析度圖形的顯示幕

1,920 個交點（ 40 行乘上 48 列），而您可以指定任何一個交點爲 16 種顏色中的一種。表 6－1 中就是顏色與代號的對照。您不必知道在 APPLE Ⅱ 中它是如何處理圖形的問題，您只需要知道如何在BASIC 語言中使用這些功能就可以了。

表 6－1 低解析度圖形的顏色

顏 色	數 字	顏 色	數 字
Black	0	Brown	8
Magenta	1	Orange	9
Dark Blue	2	Gray #2	10
Purple	3	Pink	11
Dark Green	4	Light Green	12
Gray #1	5	Yellow	13
Medium Blue	6	Aqua	14
Light Blue	7	White	15

建立圖形頁

低解析度圖形的頁佔用了文字頁兩倍的位置，當您使用 BASIC 語言時，您使用下面的敍述從顯示文字（TEXT MODE）轉移到顯示圖形（GRAPHICS MODE）。

```
GR
```

當這個敍述被執行後，顯示螢光幕上除了底部四行的位置被用來保存文字資料外，其餘的部份都變成黑色的了，底部四行的位置被稱爲文字資料幕（TEXT WINDOW），由於在螢光幕的下方存有這個文字資料幕，所以在低解析度圖形情況時，共有40行乘48列的位置供圖形使用。您可以在程式中多次的使用這個敍述，即使當您已經在低解析度圖形情況下時，您也可以使用它來清除螢光幕。

全螢幕的圖形(FULL SCREEN GRAPHICS)

在GR敍述被執行之後，您可以輸入下面的命令將文字資料幕消除使得顯示圖形的位置增多八行：

```
POKE -16302,0
```

這時文字資料幕就消失了，代之的是圖形顯示。

重新叫回文字資料幕

如果APPLE Ⅱ目前處於全螢幕圖形的情況下，您可以使用兩種方法將文字資料幕叫回。您可以使用GR敍述同時清除圖形螢光幕並把文字資料幕叫回；如果您想要不變動前面40列的圖形而將文字資料幕

叫回，那麼您必須輸入下面的敍述：

```
POKE -16301,0
```

只要這個敍述被執行了，您又可以在螢光幕的下方看到文字幕。

重返全螢幕的文字資料(FULL SCREEN TEXT)

您可以使用下面的 BASIC 敍述由低解析度圖形情況回到全螢光幕文字資料的情況：

```
TEXT
```

這個敍述將顯示的情形由圖形轉回到文字。雖然GR敍述在執行時將螢光幕清除，但是 TEXT 並不這麼做；需要記得的是，文字資料與圖形它們使用同樣的記憶體位置，當 TEXT 敍述被執行後，您可能看到的是充滿了奇怪文字的螢光幕，這是因爲APPLE Ⅱ將圖形資料變成文字資料的原因；您可以使用Esc－@，CALL－936的BASIC敍述或者在APPLESOFT 中使用HOME 命令清除螢光幕。

設計圖形的敍述

整數 BASIC 及APPLESOFT 都能夠接受低解析度圖形的命令，在螢光幕上畫一個單獨的點，改變圖形的顏色或者畫出不同長度的垂直或平行綫，這些命令只能使用在低解析度圖形的第一頁位置，如果您使用第二頁，您就無法使用這些省時的命令，也就是說您必須使用PEEK、POKE及CALL 敍述來做這些工作了。

COLOR敍述

在表6－1中我們看到每一個顏色都有對應代表的數字，而這就是您在使用 COLOR 敍述時設定顏色的數字。例如：

```
COLOR=13
```

將顏色設定爲黃色。如果您不想選擇顏色，那麼APPLE Ⅱ將自動地定黑色爲設定色，也就是 COLOR＝Ø ，雖然即使您定顏色的數字可以到255而不致造成任何錯誤，但是 COLOR 敍述只會取用您選擇數字的低位數。例如，如果您輸入 COLOR＝222 （等於十六進位的DE）；它將被認爲是COLOR＝14（等於十六進位的ØE）。

PLOT敍述

這個 BASIC 敍述在您所指定的位置上畫出一個點——實際上是一個小的長方形。

這個敍述：

```
PLOT 23,18
```

在第24列與19行交會的地方點一個小長方形，使用的是前面COLOR敍述所選擇的顏色。列的數字是在Ø與47之間，行則在Ø與39之間，如果您在 PLOT 敍述中超過了這個限度，那麼您將得到錯誤訊息，而您的程式將被停止。在任何的低解析度圖形的敍述中（除了GR之外），您可以使用運算式來代替單一的數字：

```
PLOT Y/2+12,X-4
```

一個繪圖的例子

下面這個整數 **BASIC** 的程式使用了目前提過所有的低解析度圖形的敍述。它的目的是劃出一條由左上方到右下方的對角綫。

```
10 REM  DRAW A DIAGONAL LINE
11 REM  ACROSS THE LOW-RES SCREEN
20 GR
30 COLOR= RND (16)
40 FOR Y=0 TO 39
50 PLOT Y,Y
60 NEXT Y
70 GOTO 30
```

顯示螢光幕會同圖 6 － 2 一般，只是對角綫的顏色一直在變換罷了。

您可以改變行號 30 的敍述，將這個程式改爲 **APPLESOFT** 的程式：

```
30  COLOR=  RND (16) * 16
```

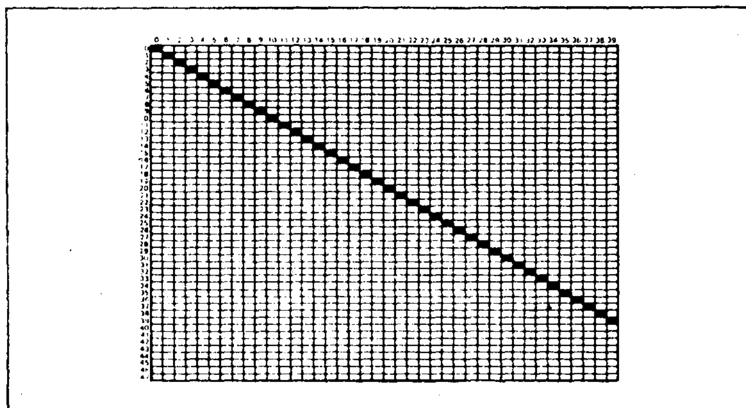


圖 6 － 2　低解析度繪圖的例子

畫水平線

HLIN 命令允許您在低解析度圖形頁上由左至右劃出不同長度的直綫。下面這個敍述：

```
HLIN 0,39 AT 0
```

劃出一條位於螢光幕頂端自最左邊到最右邊的水平直綫，HLIN 就表示水平綫（HORIZONTAL LINE），它的一般型式就是輸入左邊的行數，後面跟著逗點，接著的是右邊的行數（在上面這個例子就是39），然後是AT字，後面接著的是這一條水平綫所要劃在那一列上的數目。

左邊及右邊行數的元素不能是負值而且必須比256要小，右邊行數的值不能小於左邊行數的值，至於跟在AT後面列的數目則也不能是負值或大於 48。如果任何元素超過了它所能被允許的限度，那麼您將得錯誤訊息，而且程式就會被停止執行；如果您在 HLIN 敍述中（整數 BASIC 語言系統）指出的行數大於 39，那麼您可能得到不可預測的執行結果，至於在 APPLESOFT 中，您將得到錯誤訊息。例如，您執行下面的程式：

```
10 GR
20 COLOR=12
30 HLIN 45,100 AT 0
```

在整數 BASIC 中，您將看到兩條綠色的條狀顯示，第一個條狀顯示並不位於第 0 列——它應該處於的位置，而且這個條狀顯示繼續在螢光幕上向下方延伸數行的位置。因此在低解析度的螢光幕上將這些元素的值超出它們所能適用的範圍實在並不是一個好的主意。

劃垂直線

VLIN 敍述使用您所選擇的顏色從某列劃一條直綫到另外一列（在您所指定的行上）。

例如：

```
VLIN 12,30 AT 33
```

在行數 33 的行上自第 12 列的位置劃一條直綫到第 30 列的位置。**VLIN** 的元素值與 **HLIN** 的相同：第一個及第二個值必須在 ∅ 與 255之間，第二個元素值必須比第一個元素值爲大，而且最後一個元素值（也就是相關的行數值）必須在 ∅ 與 39 之間，如果任何元素超出了允許的範圍之外，您將在**APPLE Ⅱ** 的螢光幕上看到錯誤訊息的顯示。

在程式中使用HLIN及VLIN

下面這個程式對於 **HLIN** 及 **VLIN** 做更進一步的說明，它使用隨機的顏色劃出不同位置的直綫。您可以按下C TRL-C結束這個程式的執行：

```
 10 REM  LOW-RES HLIN AND VLIN DEMO
 20 GR : REM  USE GRAPHICS PAGE 1
 30 POKE -16302,0: REM  SET FULL-PAGE
 40 CALL -1998: REM  CLEAR ALL 48 ROWS
 50 REM  BEGIN PROGRAM
 60 COLOR= RND (16): REM  SELECT RANDOM COLOR
 70 HLIN 0, RND (40) AT RND (48)
 80 COLOR= RND (16)
 90 VLIN 0, RND (48) AT RND (40)
100 GOTO 60
```

SCRN敍述

SCRN 敍述的說明較其他的低解析度圖形的敍述稍爲難一些；假

設您想要電腦去猜想在螢光幕上的某一點位置是那一種顏色，那麼您就可以使用 **SCRN** 做這件事。下面這個敘述：

```
X=SCRN(12,24)
```

將位於括弧內座標所標示的點（也就是第 13 列與第 25 行交會處的點）的顏色的代號放入變數X內。顏色的代號是由 0 到 15 ，分別表示在表6－1 中所列的 16 種顏色。例如，如果您輸入下面這些立即式的敘述：

```
GR
COLOR=14
PLOT 12,12
PRINT SCRN (12,12)
```

那麼您將在螢光幕上看到 14 這個數字。

這個敘述在您設計較高深的低解析度圖形的程式時可能對您的幫助很大。

高解析度的圖形

APPLE Ⅱ電腦最令人覺得可取的一面就是它具有高解析度圖形的能力；如同低解析度圖形一般，您也有兩頁的記憶體位置可用，但是這只是它們唯一相似的地方。高解析度圖形的解析度是將水平部份分為280 個位置而在垂直部份分為 192 個位置，分別是低解析度圖形的 7 倍與 4 倍。雖然在高解析度圖形中您使用的顏色選擇比較少，但是却使您在螢光幕上能夠劃出更好的圖形。

在 **APPLESOFT** 中才可以使用高解析度圖形的功能，在整數 **BASIC** 中並沒有這樣的命令。然而，不論您在**APPLE Ⅱ**中使用什麼樣的程式語言，由於**APPLE Ⅱ**使用記憶體的一部份儲存高解析度的點、綫及造型而且也包含了已經被設定的程式用來在 **TV** 的螢光幕

上顯示這些記憶體的內容，它都具有顯示高解析度圖形的能力。有一些 APPLESOFT 的敍述自動地使用這些設定的程式及顯示圖形的記憶體，我們首先將討論它們；在這一節的稍後部份您將看到如何在整數 BASIC 中使用高解析度圖形的能力，當您讀到那裏時，您將發現有一些高解析度圖形的技巧只能在整數 BASIC 中被使用到。

那一頁是您應當使用的

您在使用高解析度圖形時會遇到一些困難，而這些困難又與您的 APPLE Ⅱ 所具有的記憶體數量多寡有關；如果您擁有硬體的APPLESOFT （就是存在 ROM 內或者在APPLE Ⅱ 的語言系統內），那麼如果您具有 16K 以上的記憶體，您就可以使用第一頁的位置，如果您的記憶體在 24K 以上，那麼您就可以使用第二頁，但是您若想同時使用 DOS 與 APPLESOFT 的高解析度圖形，那麼您必須另外加上 12K 的記憶體。

如果您的 APPLESOFT 是由磁帶或磁片內存入的，那麼您就不能使用高解析度的第一頁位置，因爲它是被使用來儲存部份的APPLESOFT 編譯程式，如果您想使用第一頁的位置，那麼您將損壞或是失掉使用 APPLESOFT 設計程式的能力。如果您的APPLE Ⅱ具有 24K 以上的記憶體，您就可以使用高解析度圖形的第二頁位置，或者您想要同時使用 DOS ，那麼您必須最少具有 36K 的記憶體。

設定高解析度圖形的記憶體位置

APPLESOFT 並不會自動地保護高解析度圖形的記憶體位置，所以當一個程式由於加入更多的敍述或者更多的變數，而佔用更多的記憶體位置時，毀掉圖形頁的機會就增大了。解決這個問題的方法就是設定兩個記憶體的指標，HIMEM: 及LOMEM: ，將您所使用的高解析

度圖形頁的位置保護起來，這些指標就如同邊界一般使您的程式不致於超過界限，同時使您能在這些區域內自由地使用記憶體。

HIMEM: 及**LOMEM:**的使用在 **APPLESOFT** 與整數**BASIC**中並不相同，您可以在第八章中檢查它們的差別；同樣地，在附錄G的G－1圖中也描述了記憶體的使用情形，如果您先研究過了它們之後，那麼下面的幾段您將能夠比較容易瞭解。

如果您想要使用高解析度的第一頁（您必須具有硬體的**APPLE-SOFT**），而您只具有 16K 的**APPLE Ⅱ**，那麼您必須設定**HIMEM:**爲 8191，對於**LOMEM:**則不必做任何設定，這使得您的程式只能被放在小於 8191 的記憶體位置內；這是一個嚴格的規定，但是由於您需要 8K 的位置來處理高解析度圖形，所以這個限制也是無法避免的。如果您的**APPLE Ⅱ**具有超過 16K 的記憶體，您可以只設定**LOMEM:**爲16384 而不管**HIMEM:**，這樣子會使您的**APPLE-SOFT** 程式被存在高解析度第一頁上方的位置。需要特別注意的是：除非您擁有超過 32K 的**APPLE Ⅱ**，絕不要使用 **DOS** 在上面這一種方法中，否則，您必須堅持使用第一種方法，它將使您的**APPLE-SOFT** 程式存在第一頁的下方。

如果您想要使用高解析度的第二頁（只能被使用在具有 24K 以上的機器），您可以設定**LOMEM:**爲24576，或者設定**HIMEM:**爲16383 而不改變**LOMEM:**。您可以調整**LOMEM:** 的值，使您的**APPLESOFT** 程式放在高解析度第二頁的位置之上。如果您想要同時使用**DOS**，那麼您必須具有至少 48K 的記憶體。如果您調整**HIMEM:**，那麼您的程式就存在高解析度第二頁的位置之下，在後面這種情況時，如果您使用的是由磁帶或磁片上存入的 **APPLESOFT** 編譯程式，那麼您的 **APPLESOFT** 程式將被夾在編譯程式之上而在高解析度第二頁位置之下，而這一段位置實在是很小的。

如果您想在高解析度圖形中使用兩頁的位置（只能在硬體的**APP-LESOFT** 如此做），您必須設定 **LOMEM:** 爲24576或者設定

HIMEM: 為 8191 ;如果您調整 **LOMEM:** 不理會 **HIMEM:** 而想將您的 **APPLESOFT** 程式放在高解析度頁之上，那麼您必須具有最少 48K 的記憶體位置，以便在同時使用 **DOS** 時，您仍然有足夠的位置供您的 **APPLESOFT** 程式使用。

設定圖形顯示

雖然對於高解析度圖形而言有兩頁的位置可供使用，但是由磁帶或磁片進入的 **APPLESOFT** 會使用第一頁的一部份位置儲存編譯程式。如果您擁有 **APPLE Ⅱ PLUS** 或者標準的 **APPLE Ⅱ**，不論您使用 **APPLESOFT** 語言卡或是語言系統卡，您都可以使用第一頁的位置而不會影響到 **BASIC** 語言。但是為了能與其他的 **APPLE Ⅱ** 共同使用同樣的程式，您也許希望使用高解析度第二頁的位置。

在 **APPLESOFT** 中，下面這個敘述：

HGR

先將螢光幕清除然後將高解析度的第一頁顯示出來，並在螢光幕的下方預留了四行位置的文字資料幕，您可以在這種情況下使用 **APPLE-SOFT** 的 **PRINT** 敘述在文字資料幕內顯示文字資料，使得螢光幕同時可以顯示圖形及文字資料。然而，當 **APPLE Ⅱ** 執行了 **HGR** 命令之後，螢光幕只能顯示 192 個水平行中的 160 行，您可以在 **HGR** 命令後使用 **POKE-16302**，∅命令得到全螢光幕的圖形顯示，這個命令將把文字資料幕消除，代之以圖形的另外 32 行位置。

您可以使用下面的敘述顯示出高解析度的第二頁：

HGR2

在 **HGR2** 執行後，將會使高解析度第二頁的192行全部顯示出來，而不在螢光幕的底端預留四行的文字資料幕。您可以在 **HGR2** 之後

使用POKE－16302，∅命令將文字資料幕打開，然而，在高解析度第二頁內使用文字資料幕較爲困難，因爲顯示在這個文字資料幕內的文字資料是存在第二個文字資料頁內的，BASIC 語言只能使用 POKE 敍述去處理它們（並非用 PRINT），而這樣的方法是受到很大的限制的；還有就是這第二個文字資料頁並不如第一個文字資料頁般地被保護住，使得不致被 BASIC 所佔用，所以您必須設定 LOMEM: 値爲 3071 或超過 3071。

HGR及HGR2的可變性

使用HGR 及 HGR2 命令時最令人覺得不方便的就是當它們被執行的時候，不論您喜歡或不喜歡它們都將把高解析度頁清除掉，更重要的是，它們不能被使用在整數 BASIC 語言中。但是您可以使用 PEEK、POKE及CALL敍述建立更具有伸縮性的圖形頁，而且您可能發現不論您使用何種語言，它們都是非常有用的。

建立圖形顯示的另外一種方法

對於進入高解析度圖形情況而不將螢光幕清除並不是不可能的事，您可以藉著打開一連串的開關（SWITCHES）而達成這個目的；理論上，您所要做的就是打開一組預存好的記憶體位置——被稱爲軟性開關（SOFT SWITCHES），也就是從－16304到－16297（$C050到$C057）的記憶體位置。在附錄E的圖E－1中對這些開關做了一個說明。爲了要使這些敍述同樣能在整數 BASIC 及APPLESOFT 中被使用，我們使用負整數的方式來表示這些記憶體位置（例如，－16304 代表 49231）。

您可以執行下面的敍述，使得高解析度圖形的第一頁被顯示出來而不致於將它前面所存的圖形清除掉：

POKE -16304,0	設定圖形情況
POKE -16297,0	設定高解析度圖形情況
POKE -16300,0	選定高解析度圖形第一頁（只有從第二頁跳回第一頁時才需要）

您可以在立即式情況下試著執行這些敍述。

至於想要顯示高解析度圖形的第二頁而且不使該頁從前所有的圖形被清除掉，您可以輸入下面的敍述：

POKE -16304,0	當圖形情況尚未被設定時才需要
POKE -16297,0	如果高解析度圖形情況尚未被設定才需要
POKE -16299,0	選擇高解析度的第二頁

在高解析度圖形顯示後進入正常BASIC 情況

在整數 **BASIC** 中，**TEXT** 及**GR**敍述可能無法有效地使螢光幕進入適當的顯示情況；如果您曾經使用了圖形的第二頁，您必須使用**POKE-16300，ø** 敍述重新選擇第一頁，否則您見到的將是第二頁內文字與低解析度圖形混合的記憶體內容，這使得文字資料的顯示非常混亂，這時鍵盤似乎無法輸入任何資料，因爲這時您由鍵盤輸入的任何資料都被放入第一頁的記憶體位置內，而您所看到的是第二頁記憶體內容的顯示。

在整數 **BASIC** 中，**GR**敍述並不能使圖形顯示由高解析度轉爲低解析度圖形，它只是將圖形的顯示加上下面四行的文字資料幕，您必須使用**POKE-16298，ø** 敍述將圖形顯示轉爲低解析度圖形。

將高解析度圖形頁清除

如果您在 **APPLESOFT** 中使用高解析度圖形，那麼 **HGR** 及 **HGR2** 敍述在被執行時，就會將被選擇的頁清除乾淨；然而，在整數

BASIC 中却沒有任何一個敍述可以達成這個功能。下面這個副程式就是使用了一個設定在監督程式內的功能將高解析度圖形的螢光幕清除乾淨：

```
18990 REM  ***************************
18991 REM  *    CLEAR HI-RES SCREEN    *
18992 REM  * (USES MONITOR'S "MOVE"    *
18993 REM  *  SUBROUTINE AT $FE2C      *
18994 REM  * TO MOVE DATA QUICKLY      *
18995 REM  * THROUGH THE HI-RES AREA)*
18996 REM  * -------------------------*
18997 REM  * SET PAGE=1 OR 2; THE      *
18998 REM  * ROUTINE DOES THE REST.    *
18999 REM  ***************************
19000 START=32
19010 IF PAGE=2 THEN START=64
19020 POKE 60,0
19030 POKE 61,START
19040 POKE 62,254
19050 POKE 63,START+33
19060 POKE 66,1
19070 POKE 67,START
19080 POKE -16304,0
19090 POKE -16297,0
19100 IF PAGE=1 THEN POKE -16300,0
19110 IF PAGE=2 THEN POKE -16299,0
19120 POKE START*256,0
19130 CALL -468
19140 RETURN
```

這個 BASIC 副程式用零將您所選擇的高解析度圖形頁塡滿；如果變數 PAGE 被設定爲 1，那麼這個副程式就會把第一頁淸除掉，如果 PAGE = 2，則第二頁就被淸除了。這個副程式只能在整數 BASIC 中使用；然而，您不要認爲這是非常不方便的，APPLESOFT 中的 HGR 或 HGR 2 敍述較這個副程式更爲有效。

高解析度圖形的顏色

在高解析度情況下有八種顏色可供您選擇，但只有四種不同的顏色是實際上被使用的，除了這四種之外尚有黑色與白色可供選擇。表 6－2 對於顏色與代表它們的代號做了一個對照。

表 6－2 高解析度圖形的顏色

顏 色	數 字	顏 色	數 字
Black	0	Black	4
Green	1	Orange	5
Violet	2	Blue	6
White	3	White	7

HCOLOR敍述

APPLESOFT 的 HCOLOR 敍述選擇在高解析度情形下所能顯示八種顏色其中的一個供做圖形的顯示。 HCOLOR 並不像低解析度情況下的 COLOR 敍述一般（這個敍述允許您指出大於所能代表顏色數目的數字），它只能接受小於或等於7的數字。如果您指出一個超出範圍的顏色代表數字，您的程式就會被打斷而您將得到？ILLEGAL QUANTITY ERROR 的錯誤訊息。 HCOLOR 無法將已經顯示的圖形改變顏色，並不如在低解析度圖形般的產生效果。

設定高解析度的底色

只要對前面用來清除高解析度螢光幕的副程式稍做修改，它就能將螢光幕的底色用八種顏色中的一種顯示出來了。下面這個副程式就是做這個工作，在呼叫這個程式之前，您必須先將圖形頁的代號放入 PAGE 內，同樣的，也要將代表顏色的數字放入 HCOLR 變數內；呼叫的程式首先必須先將軟性開關設定爲高解析度圖形的情況。這個副程式只能在整數 BASIC 中使用。

```
18990 REM  *********************
18991 REM  * SET BACKGRND COLOR*
18992 REM  *-------------------*
18993 REM  * SET PAGE=1 OR 2;  *
18994 REM  * ALSO SET HCOLR TO *
18995 REM  * APPLESOFT EQUIVA- *
18996 REM  * LENT HI-RES COLOR.*
18997 REM  *********************
19000 START=32
19010 IF PAGE=2 THEN START=64
19020 POKE 60,0
19030 POKE 61,START
19040 POKE 62,253
19050 POKE 63,START+33
19060 POKE 66,2
19070 POKE 67,START
19075 Y1=85:X1=42
19080 IF HCOLR>0 AND HCOLR<>4 THEN 19090:X1=0:Y1=0
19090 IF HCOLR<>3 AND HCOLR<>7 THEN 19100:X1=127:Y1=127
19100 IF HCOLR=1 OR HCOLR=5 THEN 19120
19110 X3=X1:X1=Y1:Y1=X3
19120 IF HCOLR<4 THEN 19140
19130 Y1=Y1+128:X1=X1+128
19140 POKE START*256,X1
19150 POKE START*256+1,Y1
19160 IF PAGE=1 THEN POKE -16300,0
19170 IF PAGE=2 THEN POKE -16299,0
19180 CALL -468
19190 RETURN
```

劃出點及線

在 **APPLESOFT** 高解析度圖形中一個最有效的功能就是它能夠劃出任何垂直或平行綫或者是任何角度的綫，就如同描點一般地方便。**HPLOT** 敍述可以在三方面來使用：

```
HPLOT 12,12
```

這個敍述在目前被選擇的高解析度頁的第 13 列及第 13 行的交點處使用剛被選擇的顏色劃出一個點。

HPLOT 的第二個用法是：

```
HPLOT 0,0 TO 279,191
```

這個敍述從右上角到左下角劃出一條對角線，使用像上面一般的HPLOT 敍述，您可以從螢光幕上的任何一點劃條直線到另外一點。

第三種的型式更爲複雜：

```
HPLOT 0,0 TO 279,0 TO 279,191 TO 0,191 TO 0,0
```

這種型式的 HPLOT 只能在硬體的 APPLESOFT 中被使用，由磁片或磁帶進入的 APPLESOFT 不允許這種型式的 HPLOT 。您可以簡單的使用這種方式定義多重的繪圖敍述，所有的綫段都是使用同一樣的顏色；當您在設計一個多邊型的造型時，這種型式的 HPLOT 將會非常有用。

HPLOT的改換

下面的副程式允許您在整數 BASIC 中設計高解析度圖形，而不需要更換頁的位置；然而，由於所有的計算都是在 BASIC 中執行的，這個副程式的速度很慢，但是它却是非常有用的。使用這個副程式時，將您所要繪的點的座標輸入到X及Y內，確定 BASIC 內放的是您所要使用的高解析度頁的數字；至於設定高解析度圖形及全螢光幕的圖形則全依您的需要。如果您在使用這個副程式之前沒有設定高解析度情況，那麼這個副程式將把圖形繪入高解析度圖形的記憶體內但却不改變螢光幕的顯示，稍後，在這個程式將圖形繪在記憶體內的工作完成後，就可以使用一連串的 POKE 敍述（如前面所描述的一般）將顯示改變爲高解析度圖形，而這時隱藏在畫面後的圖形就會突然的顯現出來了。這使得您可以在顯示第一頁的高解析度圖形時，同時設計第二頁的圖形；同樣的，在相反的情況下也可以這麼做。

```
20000 REM  ************************
20001 REM  * HI-RES INTEGER PLOT *
20002 REM  * ------------------- *
20003 REM  * SET X=COL, Y=ROW,    *
20004 REM  * PAGE=1 OR 2; USES    *
20005 REM  * VARS Y1,X1,X3        *
20006 REM  ************************
20007 REM
20010 Y1=PAGE*8192: REM   SET BASE ADDRESS
20020 Y1=Y1+(Y/64)*40+(Y MOD 8)*1024
20030 Y1=Y1+(Y MOD 64/8)*128+X/7
20040 X1= PEEK (Y1): REM  READ IN THE HI-RES BYTE
20050 X3=2 ^ (X MOD 7)
20060 REM  'OR' THE BYTE IN X1 WITH
20070 REM  THE BIT VALUE IN X3.
20080 IF X1 MOD (X3 ^ 2)<X3 THEN X1=X1+X3
20090 POKE Y1,X3
21000 X1= PEEK (Y1)
21010 X3=2 ^ (X MOD 7)
21020 IF X1 MOD (X3*2)<X3 THEN X1=X1+X3
21025 X3=2 ^ (X MOD 7)
21026 GOTO 21100
21030 POKE Y1,X3
21040 RETURN
21100 POKE Y1,X3
21130 RETURN
25000 GOSUB 19000
```

一個整數BASIC高解析度圖形的例子

下面這個程式利用前面介紹的兩個副程式在高解析度螢光幕上描點；遊戲控制器決定所要描的點的座標，轉動控制器的操作桿，您就可以在螢光幕上畫出綫條來了。

```
 10 REM  THIS PROGRAM USES SPECIAL
 20 REM  SUBROUTINES TO CLEAR AND
 30 REM  PLOT IN HI-RES GRAPHICS
 40 REM  USE CTRL-C TO END PROGRAM
 89 REM  SET GRAPHICS MODE
 90 POKE -16304,0
 99 REM  SET HI-RES GRAPHICS
100 POKE -16297,0
109 REM  SELECT FULL-SCREEN GRAPHICS
110 POKE -16302,0
200 PAGE=2
204 REM  CLEAR HI-RES SCREEN MEMORY
205 GOSUB 19000
209 REM  GET POINT COORDINATES
```

```
210 X= PDL (1)
220 Y= PDL (0)
229 REM  PLOT HI-RES POINT
230 GOSUB 20010
239 REM  GET MORE COORDINATES
240 GOTO 210
260 END
```

對於這個程式一個有趣的改變就是使用顯現底色的副程式代替清除畫面的副程式。

您可以試著改進這個程式，使用 POKE 敍述檢查遊戲控制器的壓點，而且在當壓點被按下時將螢光幕清除。

使用高解析度造型

APPLE Ⅱ不僅僅讓您繪出點與點間的各種圖形，同樣地它也允許您在高解析度圖形的情況下設定並處理二階的造型（ SHAPE ）。這一節將描述如何在 APPLESOFT 中建立、設計並使用一個造型，雖然如此，這一節可能只是告訴您一個開始的工作，往後更深遠的發展就全看您自己的研究了。

如果您曾經設計過繪地理圖形的高解析度圖形，您也許遇到一些如何將圖形在螢光幕上處理的困難。例如，也許您想要將圖形旋轉或者使它在螢光幕上放大或縮小。高解析度造型就具有這些功能。

造型的定義

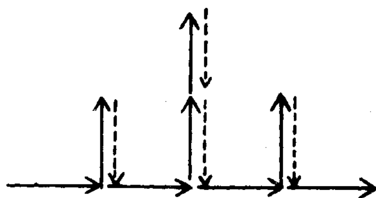
高解析度造型（ HIGH-RESOLUTION SHAPES ）需要有計劃的處理；在已往，您只是告訴電腦由A點到B點間劃一條直線，而當您在APPLE Ⅱ上使用造型時，您必須首先描述整個圖形，然後才能命令電腦將它繪出；您使用造型表（ SHAPE TABLE ）定義高解

析度造型，造型表的意義就是它包含了所要繪的圖形的特性碼。定義高解析度造型表的第一步就是先將圖型劃在紙上。下面這個例子就是繪出一個正方型，它由四條相等長度的直線所組成，每一條直線與前面所劃的直線間形成一個直角：



造型表擁有劃出圖形的指令碼，這些指令就被稱爲繪圖向量（**PLOTTING VECTORS**），每一個向量描述往上、下、左、右或者根本不移動的情形，同時也描述是否要繪在螢光幕上。您可以將上面的正方形描述爲：一上、一右、一下及一左，而這也就是**APPLESOFT** 中處理造型的程式看這個圖的方式。

如果圖形包含了對角線或是曲線，那麼就比較難以描繪了。一個三角形，雖然它只比正方形爲少一邊，但是由於它包含了至少一條的對角線，所以所費的工夫就大的多。下面圖形中的虛線表示不繪出任何圖形的移動：



由於您只能使用向量定義造型，而向量只能往上、下或左右移動，因此有些圖形，例如圓形，就不能繪的很相似，因此在某些情形下繪較複雜的圖形時，使用造型表還不如使用 **HPLOT** 來得方便。

造型表的組合

您所畫在紙上的圖形必須經過轉換變成繪圖向量，這一節中我們將告訴您如何做這個轉換的工作，下一節中介紹的是一個替您做轉換工作的APPLESOFT 程式，所以如果您對造型表內部的工作並不感興趣的話，您可以跳過這一節。

向量碼的範圍在 ø 與 7 之間，每一個定義造型的數元組可以擁有三個向量。表6 － 3中就是每一種可能的向量碼。只要造型變成了一組向量後，它就可以被放入記憶體內，這時某一些APPLESOFT 的命令就能將它們解碼（ DECODE ）然後繪出圖形。

在造型中選一個起始點，使用箭號（ ↑ ← ↓ → ）將這個造型所需要的向量列出來；向量要依順時鐘或反時鐘的方向順序列出，將必須存在但不要畫出的向量標明。如果前面正方型的造型是由左下方開始的，那麼下面就是它的向量：

方 向	繪圖否
↑	Yes
→	Yes
↓	Yes
←	Yes

現在使用表6 － 3所列的對照表將向量的對應二進位碼填入表內，下面就是您應該得到的：

係 數	碼
↑	100
→	101
↓	110
←	111

如表6 － 4所示的，造型表的每一個數元組都含有三個節（ SEC -

表6－3 繪圖係數及它們的二進位碼

符號	動作	二進位碼	十進位碼
↑	往上移動而不繪圖	000	0
→	往右移動而不繪圖	001	1
↓	往下移動而不繪圖	010	2
←	往左移動而不繪圖	011	3
↑	往上移動並繪出圖形	100	4
→	往右移動並繪出圖形	101	5
↓	往下移動並繪出圖形	110	6
←	往左移動並繪出圖形	111	7

表6－4 造型表的數元組

數元	第三段		第二段			第一段		
	7	6	5	4	3	2	1	0
M = 移動數元 P = 繪／不繪數元	M	M	P	M	M	P	M	M

TION ），每一个可能都含有一個繪圖向量。需要注意的是，第一節及第二节都各自擁有三个數元（BIT），而第三節只擁有兩個數元。

你由观察表6－3可以發現用來繪圖的碼大都是擁有三個數元的數字，這對於定義造型的數元組的第一及第二節而言是非常適當的（第一及第二節都擁有三個數元），但是對於擁有兩個數元的第三節就只能存入某些繪圖向量了；被允許存入第三節的向量只有往左、右及向下移動而且不繪出圖形這三種，其他的向量都無法存入第三節內。

大多數的時間您將發現第三節都不會被使用，這並不表示您可以將它忘記，但是在許多的情形下您可以不去管它。如果第三節被定爲ø，那麼APPLESOFT將不理會這一節，它會繼續往下一個定義造型的數元組內去讀取資料並把它翻譯繪出圖來。

繪圖向量等於ø具有兩種意義，如果第三節等於ø，那麼一個繪圖向量就表示“不移動也不繪圖”；然而，在表6－3中，一個等於ø的向量表示“往上移動但不繪圖”；這兩種不同的情形對於每一個定義造型的數元組內的第一節及第二節就造成了問題，因爲在某些情況下 **APPLESOFT** 不理會等於ø的繪圖向量，而在另一些情況下則必須往上移動而不繪圖。這裏所要告訴您的規則就是——如果您想要使用一個等於ø的向量表示“往上移動但不繪圖”，那麼絕對不要使用一個等於ø的向量做爲定義造型的數元組的結束。**APPLESOFT** 的造型處理程式假設如果第三節或第一及第二節被定爲ø，那麼對於被設定爲ø的節就不做任何的繪圖動作。

如果一個定義造型的數元組的三個節都被設定爲ø，**APPLESOFT** 就認爲這是一個“造型定義結束”的訊號。事實上，您必須使用一個結束的數元組（設定爲ø）結束定義造型的工作；否則，**APPLESOFT** 將超過您原先的造型而繼續的畫下去直到它遇上了等於ø的數元組時爲止。

只要有一個在同一數元組內不相等的向量跟在後面，您就可以任意使用“往上移動但不繪圖”這個動作。例如，若您將第二節設定爲ø（表示“往上移動但不繪圖”），而第三節被設定爲ø1、1ø或11（二進位數值），那麼第二節將被認爲“往上移動但不繪圖”；如果第三節被設定爲ø而第二節也被設爲ø，那麼就不會有任何的移動發生而 **APPLESOFT** 將尋找下一個數元組的第一節以便找到下一個有效的繪圖向量。

您有了這些知識之後，現在可以將造型的每一段安排成一組組的二進位的繪圖向量了。使用這種方法，您就可以將具有三個數元的繪圖向量碼組成具有八個數元的數元組，而這些數元組就可以被存在記憶體內。對於正方型的繪圖向量的組合就在表6－5中。

使用二進位碼的數元組所表示的造型，您可以很容易地將它們轉換成十六進位的表示方式。在附錄J中包含有二進位對十六進位的轉換

表6－5 建立一個造型表的解碼部份（正方的造型）

	向量			二進位碼			十六進位碼
	第一段	第二段	第三段	第一段	第二段	第三段	
數元組 0	不動作	↑	→	00	100	101	25
數元組 1	不動作	↓	←	00	110	111	37
數元組 2	不動作	不動作	不動作	00	000	000	00
數元組 3	—	—	—	—	—	—	—

表。表6－5中同樣的對正方型的十六進位碼做了轉換的工作。

現在這個造型的定義已經完成了，下一步驟就是去建立這個造型（及其他造型最多可到255個的造型）的一連串的指標，**APPLESOFT** 使用這些指標做爲索引檔。

造型表索引檔的組成

一個造型表的索引檔就是一連串的數元組，它被用來描述在這個表中具有多少的造型，同時也指到每一個造型定義在表中的位置。索引檔的第一個數元組存有表中所有造型的數目，這個數目在 ∅ 與 255 之間，第二個數元組不被使用而且應該被設定爲 ∅。

在索引檔內其他的數元組則包含了存在表中每一個造型定義的指標，每一個指標佔用兩個數元組的長度，而且存在由索引檔起始處到這個造型的絕對距離（以數元組爲單位），指標的低次（**LOW-ORDER**）數元組在高次（**HIGH-ORDER**）數元組的前面。例如，若這表的絕對距離是10個數元組，那麼您將這個指標編爲十六進位的0A00；在我們這個正方型的例子中，索引檔中只有這一個造型，所以這個造型1的絕對距離就是4個數元組，因此0400就被移入這個索引檔的第一節內。

數元組

0	01	← 在表中的造型數目
1	02	
2	04	} ← 第一個造型距離第 ϕ 個數元組的絕對距離（低次位放在前面）
3	00	
4	25	} ← 造型定義
5	37	
6	00	← 表示造型的結束

對於在造型表的索引檔結束處多留一些額外的數元組以便爲以後的造型表的指標預留空位是一個很好的習慣；如果您在索引檔的後面沒有預留空位，那麼在您要插入新的造型指標時，您必須重新安排整個的造型表，即使您可能只需要一個擁有十個造型的索引檔，您也要在它的後面留下不被使用的空位；額外的 20 個數元組就可以供您在稍後再增加 10 個造型指標了。當您想要加入其他的造型，那麼您必須將這新的造型定義放在表中最後一個造型定義的後面，計算這個新造型的絕對距離，將這新的指標放在索引檔內最後一個造型指標的後面，同時並在索引檔的第 ϕ 個數元組內加上 1（第 ϕ 個數元組內存有表內所擁有的造型的數目）。圖 6 − 3 就是造型表及它們的索引檔在記憶體內的組織情形。

使用電腦組成係數

下面的這個程式是用 **APPLESOFT** 語言設計的，替您做造型定義的組合工作；這個程式將要求您輸入每一個繪圖向量而且決定是否要將它繪出，在輸入最後一個向量之後，您必須輸入E代表“結束”並按下RETURN 鍵；然後這個程式會詢問您那一個向量是要改變的，如果您在輸入時犯了任何錯誤，那麼您就可以輸入代表這個向量的號碼並做重新輸入的工作。

如果您沒有任何需要修改的地方，那麼您對VECTOR TO CHANGE（ϕ＝END ）這個詢問輸入 ϕ；在幾秒鐘後，繪圖向量就

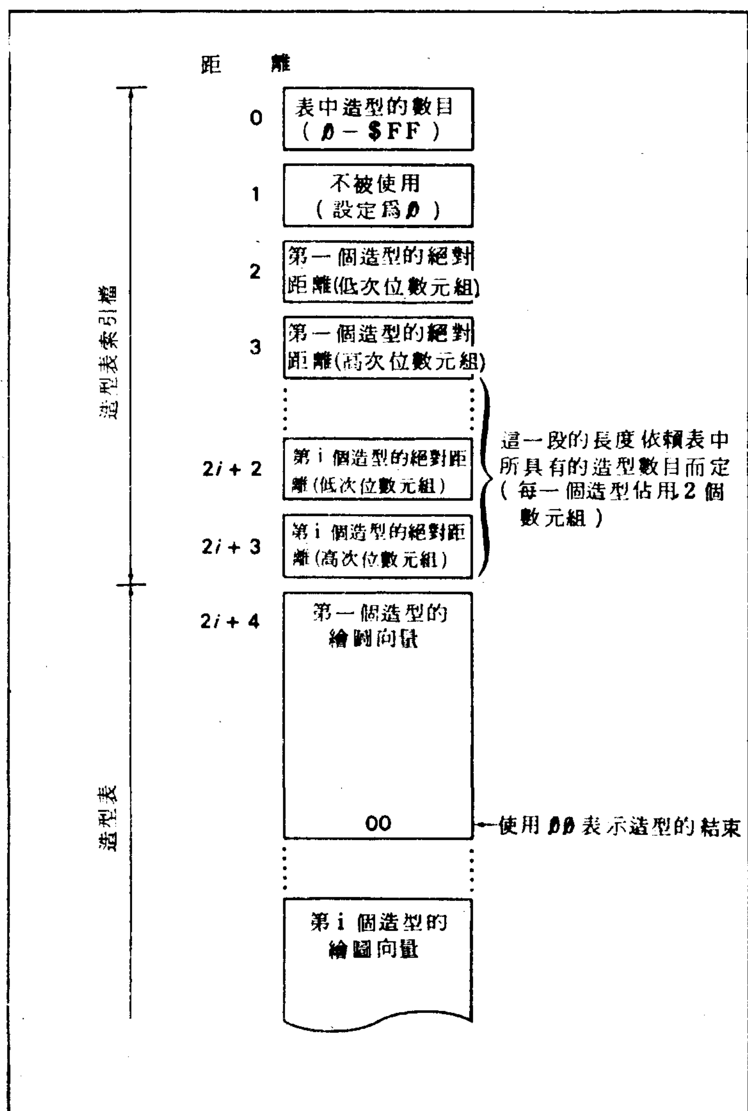

距　離
0
表中造型的數目
(∅－$FF)
1
不被使用
(設定爲∅)
2
第一個造型的絕對距離(低次位數元組)
3
第一個造型的絕對距離(高次位數元組)
2i＋2
第i個造型的絕對距離(低次位數元組)
2i＋3
第i個造型的絕對距離(高次位數元組)
這一段的長度依賴表中所具有的造型數目而定(每一個造型佔用2個數元組)
造型表索引檔
2i＋4
第一個造型的繪圖向量
00
使用∅∅表示造型的結束
第i個造型的繪圖向量
造型表

圖6－3　造型表的構造

會以十六進位的表示方式顯示出來。下面就是這個程式的程式列以及一個執行的例子：

```
1  REM  **************************
2  REM  * SHAPE CREATION PROGRAM *
3  REM  *                        *
4  REM  **************************
10  DIM S1(100),V1(100)
20 I = 0
30  PRINT "CREATE SHAPE VECTORS"
40  PRINT
41  REM  ENTER PLOT ACTIONS
50 V = I: GOSUB 270
58  REM  CONTINUE ENTRY UNTIL M$
59  REM  EQUALS TERMINAL VALUE "E"
60  IF M$ < > "E" THEN S1(I) = M:I = I + 1: GOTO 50
70  PRINT
71  REM  ALLOW CORRECTIONS
80  INPUT "VECTOR TO CHANGE (0=END):";V
90  IF V > 0 THEN V = V - 1: GOSUB 270:S1(V) = M: GOTO 80
99  REM  PACK VECTORS INTO V1() ARRAY
100  FOR V = 0 TO I
110  IF B = 2 AND S1(V) > 0 AND S1(V) < 4 THEN 140
120  IF B < 2 AND (S1(V) > 0 OR S1(V) > 4) THEN 140
130 B = 0:Q = Q + 1
140 V1(Q) = V1(Q) + S1(V) * (8 ^ B)
150 B = B + 1
160  IF B > 2 THEN B = 0:Q = Q + 1
170  NEXT V
178  REM  DISPLAY THE VECTORS AS
179  REM  HEXADECIMAL NUMBERS
180  PRINT "BYTE","VECTOR"
190  FOR V = 0 TO Q
200 H% = V1(V) / 16
210 L% = V1(V) - H% * 16
220  IF H% > 10 THEN H% = H% + 7
230  IF L% > 10 THEN L% = L% + 7
240  PRINT V, CHR$ (H% + 176); CHR$ (L% + 176)
250  NEXT V
260  END
269  REM  VECTOR INPUT SUBROUTINE
270  PRINT "VECTOR ";V + 1;":";
280  INPUT "MOVE: U/D/L/R? ";M$
290 M = 0
300  IF M$ = "R" THEN M = 1
310  IF M$ = "D" THEN M = 2
320  IF M$ = "L" THEN M = 3
330  IF M$ = "E" THEN  RETURN
340  INPUT "PLOT (Y=YES,N=NO)? ";P$
350  IF P$ = "Y" THEN M = M + 4: RETURN
360  IF P$ = "N" THEN  RETURN
370  GOTO 340
```

```
]RUN
CREATE SHAPE VECTORS
VECTOR 1:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? N
VECTOR 2:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? N
VECTOR 3:MOVE: U/D/L/R? L
PLOT (Y=YES,N=NO)? Y
VECTOR 4:MOVE: U/D/L/R? L
PLOT (Y=YES,N=NO)? Y
VECTOR 5:MOVE: U/D/L/R? U
PLOT (Y=YES,N=NO)? N
VECTOR 6:MOVE: U/D/L/R? U
PLOT (Y=YES,N=NO)? Y
VECTOR 7:MOVE: U/D/L/R? U
PLOT (Y=YES,N=NO)? Y
VECTOR 8:MOVE: U/D/L/R? U
PLOT (Y=YES,N=NO)? Y
VECTOR 9:MOVE: U/D/L/R? R
PLOT (Y=YES,N=NO)? N
VECTOR 10:MOVE: U/D/L/R? R
PLOT (Y=YES,N=NO)? Y
VECTOR 11:MOVE: U/D/L/R? R
PLOT (Y=YES,N=NO)? Y
VECTOR 12:MOVE: U/D/L/R? R
PLOT (Y=YES,N=NO)? Y
VECTOR 13:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? N
VECTOR 14:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? Y
VECTOR 15:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? Y
VECTOR 16:MOVE: U/D/L/R? D
PLOT (Y=YES,N=NO)? Y
VECTOR 17:MOVE: U/D/L/R? L
PLOT (Y=YES,N=NO)? N
VECTOR 18:MOVE: U/D/L/R? L
PLOT (Y=YES,N=NO)? Y
VECTOR 19:MOVE: U/D/L/R? E

VECTOR TO CHANGE (0=END):0
BYTE                VECTOR
0                   12
1                   3F
2                   20
3                   64
4                   2D
5                   15
6                   36
7                   1E
8                   07
9                   00

]
```

輸入造型表

在您能夠將造型顯示出來之前，您必須先將造型碼輸入到電腦的記憶體內。爲了要做這件事，您必須決定造型表應該存在記憶體的那一個區域內，最簡單的方法就是重新設定 HIMEM: 指標的值，使得在 DOS 的起始位置之下，或者在您所要使用的高解析度頁的位置之前。在您執行 APPLESOFT 的敍述使用字串之前您必須再重新設定 HIMEM: 的值。如果您使用的是由磁碟進入的 APPLESOFT ，那麼您最少需要 36K 的記憶體（ 48K 也許更好）來容納APPLE-SOFT 的編譯程式及 DOS 。位置115及116（ $73 及 $74 ）的記憶體內存著爲 APPLESOFT 設定的 HIMEM: 位置，這個位置是以低次位數元組放在前面的方式儲存的。您可以使用下面的敍述計算允許放入造型表的 HIMEM: 值：

```
PRINT PEEK(116) * 256 + PEEK(115) - X
```

這個敍述計算出應該設定的 HIMEM: 值，元素X就是造型表（包含索引橘）的長度；使用這個敍述，X用造型表的長度代替，將 HIMEM: 值在輸入造型表到記憶體內之前計算出來，這麼做就將造型表保護住使不致被APPLESOFT 所毀壞。

您也可以將造型表放在記憶體位置768與975（ $300及$3CF ）之間，但是必須要注意的是造型表所在的位置不能與機器語言的程式或您的 APPLESOFT 程式有衝突的地方。

您可以使用 POKE 敍述將造型表放入記憶體內。例如，下面一連串的敍述將我們前面正方形的造型表放在由768開始的記憶體位置内：

```
]POKE 768,01
]POKE 769,00
]POKE 770,04
]POKE 771,00
]POKE 772,37
]POKE 773,55
]POKE 774,00
```

您也可以由監督程式內輸入造型表，使用CALL－151的敍述進入監督程式，然後輸入造型表開始的十六進位的記憶體位置，後面跟著冒號，再輸入造型表內索引檔的第一個數元組。輸入一個空白，再輸入索引檔的下一個數元組及一個空白，繼續將索引檔輸入完畢，按下RETUR鍵；現在再輸入另外一個冒號後面跟著第一個造型表，每一個數元組的十六進位資料，每一個造型表後面都跟著一個空白，然後按下RETURN鍵；對於所有在索引檔內的造型重複最後一個步驟。您可以輸入十六進位的起始記憶體位置，接著一個句號，然後是造型表的結束位置，再按RETURN鍵，那麼您就可以察看您的作品了。您可以參考第7章以便瞭解更多的監督程式的功用。下面就是您在監督程式內輸入正方形的造型表的順序：

]CALL -151	
*	
*6000:01 00 04 00	監督系統標示
*:2C 3E 00	直接輸入造型表
*6000.6006	輸入第1個造型
6000- 01 00 04 00 2C 3E 00	顯示記憶體內資料做檢查的工作

現在這個造型表已經存在記憶體內了，第一個輸入就是由造型表的起始位置開始（在這個例子中是$6ØØØ），冒號（：）告訴監督程式將其後一連串的十六進位的數字放入記憶體內，在冒號之後緊跟著的就是

造型表索引檔：ø1（在造型表中造型的數目），øø（第二個不被使用的數元組），ø4 及 øø（造型 1 的絕對距離，低次位數元組放在前面）；下一行由一個冒號開始，如果您的開始位置是跟隨在前面位置的後面，那麼您不必指出這個開始位置，監督程式會把隨後一連串的十六進位的數字放在前面輸入的一串數字的後面。

最後一行的敍述告訴監督程式將記憶體位置$ 6000到$ 6006 內的內容顯示出來；這個命令的格式是：起始位置，後面跟著句點（用來告訴監督系統將記憶體內容顯示出來），再接著結束位置。在您每一次使用監督程式輸入造型表時，您必須仔細地檢查每一個輸入。

將造型表存在磁碟或磁帶內

如果您花了許多的時間設計了一個造型表，那麼將它存在磁帶或磁碟內總比在關機時就失掉它來得好得多了吧！使用磁碟一直是一個較好的方法，因爲它記錄或收取資料的速度較磁帶都快得多。

在 **APPLESOFT** 中有一個專爲磁帶使用的命令 **SHLOAD** 。在您將造型表存在磁帶中之前，您必須計算表的十六進位長度。我們仍然使用前面例子的正方形，計算一下它的造型表所佔的數元組數目是7；那麼使用監督程式，將這個長度以兩個數元組的十六進位的數字輸入由ø開始的起始位置內：

```
*00:07 00
```

如同往常的情況一般，低次位數元組放在前面，將磁帶轉回到起始處，然後按下 **RECORD** 鍵，您就可以使用監督程式將造型表及它的長度記錄在磁帶內了。這個工作必須使用兩個寫的動作，兩者也可以由同一行內輸入：

```
*0.1W 6000.6006W
```

第一個寫的命令將您輸入的造型表的長度輸入到磁帶內，第二個寫的命令則將造型表所佔的七個數元組存入磁帶內（自$6000到$6006 位置內的資料），這兩個寫的動作大約費時 20 秒左右，當記錄的工作進行時，APPLE Ⅱ的發聲器會發出兩聲“嗶”的聲音，在第二聲“嗶”發出以後，您就可以停止錄音機的轉動了；現在這個造型表已經被存在磁帶內了。在您實際操作的時候，您就用實際造型表的長度、起始位置及結束位置代替這個例子中的數字。

使用磁碟存造型表，您首先進入 APPLESOFT 的DOS 情況下（由監督程式內，您輸入 3D∅G 即可）；您使用 BSAVE 命令存造型表，使用這個命令之前您必須知道輸入的造型表的起始位置及它的長度。使用下面的 DOS 命令將從 $6000 位置開始的造型表存入磁碟內:

```
BSAVE SQUARE, A$6000, L7
```

這個命令建立了一個叫做 SQUARE 的磁碟檔，該檔的型式是B也就是二進位資料。現在這個造型表已經被存在磁碟內供您以後使用了。在您實際操作的時候，您使用實際造型表的長度、起始位置及造型表的名稱代替上面例子中的元素。

由磁帶或磁碟內將造型表存入記憶體

如果您已經把一個造型表存到磁帶內了，APPLESOFT 可以將它再讀回記憶體中。首先，將磁帶轉到起始處，再打入：

```
SHLOAD
```

在磁帶機上壓下 PLAY 鍵，APPLE Ⅱ將發出兩聲“嗶”的聲音，然後將電腦的控制轉回到鍵盤，如果在這過程中有任何不正確的情形發生，那麼您將在螢光幕上看到 ERR 的錯誤訊息；如果您看到了錯誤訊息，您就得重新讀過一遍，如果錯誤仍然存在，您就必須依照第二

章所描述的，重新設定音量的大小。 SHLOAD 命令自動地設定 HIMEM: 爲現在的 HIMEM: 值減去造型表所佔數元組的數目的差，您必須確定的是 HIMEM: 的值是正確的。

如果您是將造型表存在磁片上，那麼在您將造型表讀入記憶體之前先設定 HIMEM: 值。下面的命令將造型表由磁碟讀入記憶體內：

```
BLOAD SQUARE
```

DOS 會把您存入磁碟時的位置（在我們的例子中就是 $6000）記憶起來，並且自動地把造型表存入記憶體內的這個位置。如果您想將讀入的造型表存在另外一個位置，假設 $3000，那麼您就輸入：

```
BLOAD SQUARE, A$3000
```

在您實際使用時，您就再一次的使用確實的名字及可省略的起始位置代替例子中的元素。

在您使用 BLOAD 將造型表存入記憶體內後，您必須將造型表所在的位置存入 232 及 233（$E8 及 $E9）的位置內，APPLESOFT 使用這兩個位置指出在記憶體內造型表的位置（SHLOAD 自動地設定這個指標）。進入監督程式同時設定這個位置（例子中的 $6000）：

```
E8:00 60
```

您也可以使用 POKE 敍述將您的造型表的起始位置存到 232及233的位置去；需要注意的是，在您使用 POKE 敍述時，您必須使用十進位的值。

現在您已經可以在 APPLESOFT 程式中使用造型表了。

繪出造型的命令

APPLESOFT 有四個處理造型的敍述能夠繪出、消除及改變造

型的方向：

DRAW ，將造型顯示在**APPLE Ⅱ**的螢光幕上。
XDRAW ，將已顯示的造型消除掉。
ROT ，將造型在**X**、**Y**區域內旋轉。
SCALE ，改變已經被繪出造型的大小。

這些處理造型的命令使用您剛剛選擇過的高解析度的頁（使用**HGR**或**HGR2**）及顏色（使用 **HCOLOR** ）。

SCALE命令

在一個程式中第一次繪出造型之前，您必須在立即式或間接式的情況下，先使用 **SCALE** 命令：

```
SCALE=1
```

將設定每一個繪圖向量的範圍爲一個點，如果 **SCALE**= 5 ，那麼**APPLE Ⅱ** 將對每一個繪圖向量畫出五個點，您可以將 **SCALE** 設到 255（也就是對每一個向量繪出 255個點），最大的範圍設定爲**SCALE=ø** ，它將使每一個向量繪出 256 個點。

DRAW命令

DRAW 繪出在造型表內的造型（由 1 到 255的數字表示），使用的是最後被選擇的顏色，並依照最後設定的大小及旋轉值繪出造型。下面這個敍述：

```
DRAW 1 AT 140,96
```

將造型表內的第一個造型開始繪在高解析度螢光幕的第 141 行與第 97 列交會的位置，繪圖所在的起始位置必須在敍述中說明。第二種DRAW的選擇使用已經被設定的位置。

```
DRAW 11
```

這個敍述將存在造型表內第 11 個的造型依照最近使用 HPLOT 或 DRAW 敍述設定的位置繪出。如果座標在前面也沒有被設定過，那麼 APPLESOFT 將設定 ∅ 爲座標值。

特別要注意的是，APPLESOFT 假設造型表已經被存在記憶體內了，在您執行 DRAW 敍述之前，您必須確定造型表已經存在記憶體內，而且 232 及 233（$E8 及 $E9）的記憶體位置內存著造型表的起始位置。如果您使用了一個造型數字超出了造型表內擁有的造型數目，或者 DRAW 敍述使用的行及列的座標是超出範圍的，那麼繪圖的動作將不會發生；反之，您將得到 ?ILLEGAL QUANTITY ERROR 的錯誤訊息。

XDRAW 命令

這個敍述允許您在不改變高解析度底圖的情況下，消除一個造型。下面這個例子：

```
XDRAW 8 AT 90,96
```

這個敍述的語法與 DRAW 相同，繪圖的座標可以依上面的方式指出或者依前面的 DRAW 敍述行事。 XDRAW 檢查這個造型的顏色，然後繪出一個與這個顏色互成補色的圖形。上面這個例子， XDRAW 發生在第 91 列及第 97 行的位置。表 6 － 6 列出了所有高解析度顏色的補色。

如果座標、旋轉向量及大小都與在螢光幕上的造型相同，那麼

表6－6　繪圖顏色

如果顏色是	XDRAW的顏色是
Black	White
White	Black
Violet	Green
Orange	Blue
Green	Violet
Blue	Orange

XDRAW 就把這個圖形消除，但周圍的其他圖形仍然不改變。

ROT命令

ROT 命令依照原先螢光幕中的點來旋轉造型；下面這個敘述：

```
ROT=16
```

將造型旋轉的角度設爲順時鐘 90°，**ROT** 的值必須在 ∅ 與 255 之間，但是只有 64 種可能的旋轉係數——∅ 到 63。圖6－4顯示的就是依照 **ROT** 值圖形位置的變化情形。

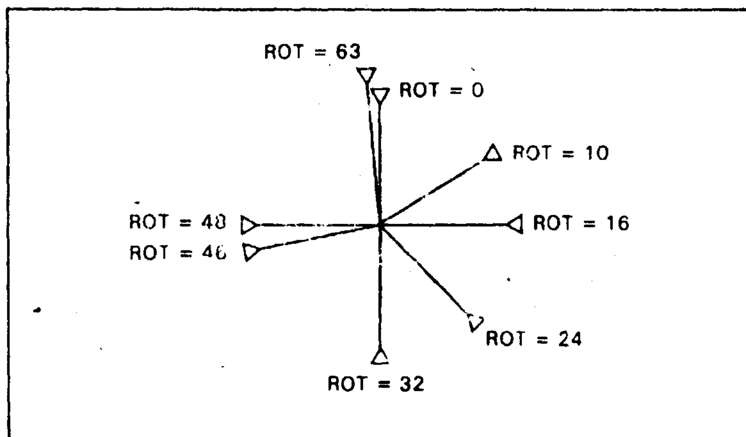
當 **SCALE**＝1 時，**ROT** 只能依 90° 的增加方式旋轉造型，也就是說只有四種有意義的造型旋轉：∅＝∅°、16＝90°、32＝180°、48＝270°。**APPLESOFT** 將會自動地把您設定的旋轉係數定爲下一個 **ROT** 所能表示的值。如果您將 **SCALE** 設爲5或較5爲大，那麼 64 種的旋轉角度都是可以被使用的。

在一個程式中使用造型

下面的程式列使用造型並且對於如何在程式中使用它們做了一個示範：

```
1   LOMEM: 24600
20   HGR2
50   REM   SET SHAPE TABLE START ADDRESS
60   POKE 232,0
70   POKE 233,96
75   REM   USE ALL COLORS IN TURN
76   FOR H = 1 TO 7
79   REM   INCREMENT ROTATION FACTOR
80   FOR I = 1 TO 80
82   HCOLOR= H
90   ROT= I
92   IF I < 50 THEN  SCALE= I
105   DRAW 1 AT 140,96
106   ROT= I + 32
110   DRAW 1 AT 140,96
130   NEXT I
140   NEXT H
150   GOTO 76
```

圖 6－4　造型的旋轉



在遊戲的程式中與高解析度造型一起使用是特別的有用。

您可以建立一個造型庫，而不必如使用 **HPLOT** 般地記錄它們；造型在記憶體中按順序排列，也節省您許多計算大小、旋轉等等的工作。如果您在處理高解析度造型時發生困難，那可能的原因是您沒有試著去設計您自己的造型；由於即使是一個簡單的造型也必須經過許多的步

驟才能達成目的，所以這件工作是變複雜的，但是如果您使用前面介紹的程式，它會自動地替您做造型的解碼工作，那將爲您省下不少的時間。

APPLE Ⅱ 的聲音

在您讀過本章前幾節的部份之後，您再研究下面關於APPLE Ⅱ發聲的部份可能使您覺得雖然輕鬆但却有一些小小的煩惱。您覺得輕鬆是因爲操作發聲器實在是一件簡單的工作，但是您却無法使用BASIC敍述控制發聲器。您必須詳細地設定每一個APPLE Ⅱ所發出的聲音；事實上，您在程式中所能做的事只是讓發聲器發出一個喀嗒的聲音；其實使用發聲器使它發出一連串的聲音的方法就是改變喀嗒聲的頻率，因此而建立了不同調子的音樂。這一節將告訴您如何操作發聲器，同時也介紹一個能夠產生一連串聲音的程式。

操作發聲器

APPLE Ⅱ使用－16338的記憶體位置，也就是49200及$C030的記憶體位置，做爲一個開關，就與前面討論的圖形的開關相似。任何時間只要您處理這個位置，就會由APPLE Ⅱ內的發聲器內發出一聲喀嗒聲。在 BASIC 語言中，您可以使用下面的敍述操作發聲器：

```
    A=PEEK(49200)
或
    A=PEEK(-16336)
```

使用 BASIC 程式操縱發聲器（下面簡單的程式）是非常簡單的，但是發聲器只會產生低頻率的聲音，這是因爲 BASIC 語言比較起來要慢得多；在整數 BASIC 中，最高的頻率可達256Hz，在APPLE SOFT 中則只能達到72Hz，唯一解決這個問題的方法就是使用一個機器語言的副程式來操縱發聲器，才能達到更高頻率的聲音。

```
 9 REM  CLICK THE SPEAKER
10 A= PEEK (-16336)
20 GOTO 10
```

機器語言發聲的副程式

APPLESOFT 及整數 **BASIC** 在記憶體位置 768 與 975（$300 及 $3CF）之間會預留機器語言副程式及造型表的位置，而不必改變 **LOMEM:** 的設定。（**DOS** 會使用這個區域，當您將早於 3.3 型式的 **DOS** 的"**SYSTEM MASTER DISKETTE**" **BOOT** 的時候，它將會佔用這個區域，而且將前面所存的資料全部毀掉）。

如果您還沒有研究過組合語言及機器語言，您仍然可以使用下面的副程式與整數 **BASIC** 或 **APPLESOFT** 程式合併使用。要將這個副程式放入記憶體內，必須輸入 **CALL-151** 的命令或者按下 **RESET** 鍵進入監督程式內（需要特別注意的是，如果您擁有 **DOS**，那麼您必須在進入監督程式之前先將它存入記憶體內），然後，您就可以輸入下面的機器語言副程式了：

```
>CALL -151

*F666G

!302:LDY 301

0302-    AC 01 03     LDY    $0301
! LDX 301

0305-    AE 01 03     LDX    $0301
! LDA #4

0308-    A9 04        LDA    #$04
! JSR FCA8

030A-    20 A8 FC     JSR    $FCA8
! LDA C030
```

電腦將您輸入的命令加上陰影顯現

```
030D-    AD 30 C0    LDA    $C030
! INX

0310-    E8          INX
! BNE 310

0311-    D0 FD       BNE    $0310
! DEY

0313-    88          DEY
! BNE 305

0314-    D0 EF       BNE    $0305
! DEC 300

0316-    CE 00 03    DEC    $0300
! BNE 302

0319-    D0 E7       BNE    $0302
! RTS

031B-    60          RTS
```

這個副程式使用＄300及＄301的位置儲存資料，＄302到＄31B的位置內則存放這個副程式；只要您輸入了這個副程式，您就可以使用下面的方法檢查它是否正確：

```
!$FF69G

*302L

0302-    AC 01 03    LDY    $0301
0305-    AE 01 03    LDX    $0301
0308-    A9 04       LDA    #$04
030A-    20 A8 FC    JSR    $FCA8
030D-    AD 30 C0    LDA    $C030
0310-    E8          INX
0311-    D0 FD       BNE    $0310
0313-    88          DEY
0314-    D0 EF       BNE    $0305
0316-    CE 00 03    DEC    $0300
0319-    D0 E7       BNE    $0302
031B-    60          RTS
031C-    20 20 70    JSR    $7020
031F-    08          PHP
0320-    18          CLC
0321-    D8          CLD
0322-    88          DEY
0323-    08          PHP
0324-    A0 A0       LDY    #$A0
0326-    10 38       BPL    $0360
* ←—使用 CTRL-B 進入 BASIC
```

如果在您的APPLE Ⅱ內已經存了DOS，那麼您可以使用 BASIC 中的 BSAVE 命令將這個副程式存到磁片上去：

```
BSAVE SOUND,A$302,L26
```

這個命令為您建立了一個叫做 SOUND 的二進位檔，在稍後您需要使用它時，可以隨時將它再存入記憶體內。如果您擁有磁帶機，那麼您輸入下面的監督程式的命令將這個副程式記錄起來：

```
*302.31BW
```

這個副程式就被存在磁帶上供您以後的使用了。

BASIC介面

您必須有一個與前面這個機器語言副程式交通的 BASIC 副程式。您輸入下面的 APPLESOFT 或是整數 BASIC 的程式列：

```
3200  REM  SPEAKER DRIVER
3210  POKE 768,D
3220  POKE 769,F
3230  CALL 770
3240  RETURN
```

這個副程式與機器語言的副程式合用時，一個 BASIC 程式可以設定兩個變數值而產生聲音：F（表示頻率），可以在1與 255之間，255就是最高的音調了；D（表示持續時間），也是可以在 1 與 255 間的數值，255 就是最長的持續時間。在輸入上面這個副程式後，再輸入下面的 BASIC 程式做為一個例子：

```
10  FOR I = 1 TO 254
20 F = 1
30 D = I
40  GOSUB 3200
50  NEXT I
60  END
```

當您執行這個程式時，對於每一個聲音您都聽聽它的持續時間，在低頻率及高頻率的結尾，聲音都較中等頻率來得短，這個問題是無法避免的，因爲機器語言副程式是使用指令而不是經由時間的改變來修正每個單音的頻率的；由於APPLE Ⅱ本身並不具有時鐘或計時器，所以這種方式是必須的；您可以將極高或極低的單音設定較長的持續時間，避免這種不平衡的情形。

一個較複雜的聲音程式

下面這個程式使用了前面的副程式建立一連串的聲音，供您聽、改變或最後將它們印出供以後其他 BASIC 程式的使用。當您執行這個程式時，您將看到(E)NTER、(L)ISTEN、(P)RINT?的訊息，第一步驟就是輸入一些音調，所以您輸入E字按下RETURN鍵。

現在您將看到TONE ∅ : FREQUENCY，DURATION?的訊息，輸入用逗號分開的兩個數字，第一個就是頻率，第二個則是持續時間，兩者都必須是在1與255之間的數字，當您輸入這兩個值並且按下RETURN鍵之後，APPLE Ⅱ的發聲器就送出這個音調，這個過程對於一連串的音調重複執行（最多可至100種音調），在您輸入最後一個音調後，打入∅的頻率及∅的持續時間結束輸入的工作。

在您輸入音調的工作結束後，WHICH TONE TO CHANGE的訊息就顯示出來了，如果您想改變您輸入的任何音調，您就打入這個音調的數字，否則，您輸入∅就可以了。

在您修正完畢後，您將再看到(E)NTER、(L)ISTEN、(P)RINT?的訊息，現在您輸入L並按下RETURN鍵，這時APPLE Ⅱ將把您輸入的一連串音調重新發出來，在最後一個音調發聲完畢後，這個訊息又再度出現，這時您輸入P顯示出或印出每一個音調的頻率及持續時間。

下面這個程式可以簡單地修正爲將這些音調的值存在磁帶或磁片上

，然後其他的程式就可以使用這些音調的值及發聲的副程式作曲或只是發出任意的聲音。

```
10   REM  SOUND GENERATOR PROGRAM
19   REM  ARRAY REMEBERS ENTERED TONES
20   DIM A(100,2)
29   REM  CLEAR DISPLAY
30   CALL  - 936
40   INPUT "(E)NTER, (L)ISTEN, (P)RINT?";A$
50   IF A$ = "L" THEN 1000
60   IF A$ = "P" THEN 1200
80   IF A$ <  > "E" THEN 30
81   REM  ENTER EACH TONE
90   PRINT
100 I = 0
105 M = I
110  GOSUB 3000
119  REM  END OF TONE ENTRY?
120  IF F = 0 AND D = 0 THEN I = I - 1: GOTO 200
129  REM  NO--REMEMBER TONE
130 A(I,1) = F:A(I,2) = D
140 I = I + 1
150  GOTO 105
200  REM  CHANGE ANY ITEMS HERE
205  PRINT "WHICH NOTE TO CHANGE (0-";I;")";
206  INPUT E
208  IF E = 0 THEN 30
210  IF E < 1 OR E > I THEN 210
220 M = E: GOSUB 3000
230 A(E,1) = F:A(E,2) = D: GOTO 205
1000  REM  LISTEN TO THE NOTES SO FAR
1010  FOR K = 0 TO I
1020 F = A(K,1):D = A(K,2): GOSUB 3200
1030  NEXT K
1040  GOTO 30
1200  REM  PRINT OUT THE NOTES
1210  PRINT "NOTE#","FREQ","DURATION"
1220  FOR K = 0 TO I
1230  PRINT K,A(K,1),A(K,2)
1240  NEXT K
1250  PRINT
1255  PRINT "PRESS RETURN TO CONTINUE": INPUT Z$
1260  GOTO 30
3000  PRINT "TONE ";M;
3010  INPUT " ENTER FREQUENCY, DURATION";F,D
3015  IF F = 0 AND D = 0 THEN  RETURN
3020  IF (F < 0 OR F > 255) OR (D < 1 OR D > 255) THEN 3010
3030  GOSUB 3200
3040  RETURN
3200  REM  SPEAKER DRIVER
3210  POKE 768,D
3220  POKE 769,F
3230  CALL 770
3240  RETURN
```

7



機器語言的監督系統

永遠存在APPLE Ⅱ ROM 內的是一個控制程式，也就是監督系統（MONITOR），這一章描述監督系統的一些功能並且教您如何將它們與您所設計的BASIC語言連接在一起使用；監督系統是使用機器語言設計的，機器語言是做爲BASIC語言（以及其他在APPLE Ⅱ所能使用的程式語言）與機器提供的一些功能（例如印出一個字，給出一條直綫等等）中間的一個連接工具。

您也可以使用鍵盤輸入的命令操作監督系統，這麼做可以使您建立圖形造型表（在第六章中描述過），檢查記憶體硬體的問題，或是使用組合語言設計程式。大多數的時間您都不需要去使用它，但是有時候監督系統具有的一些功能使您從事一些工作時更爲方便。

在描述監督系統的功能及它本身之後，這一章將告訴您如何使用MINI-ASSEMBLER 語言，這一章並不教您如何設計組合語言程式，在附錄K中的一些書會做這方面的工作。您將學到的是如何將您的組合語言與BASIC語言程式合併使用，以及如何使用MINI-ASSEMBLER設計、測試及偵錯您的組合語言程式。

監督系統的處理

監督系統具有兩個型式：標準型式（STANDARD VERSION）及自動啓動型式（AUTOSTART VERSION）。如果您的APP-LE Ⅱ具有標準型式的監督系統，只要將電腦的開關打開您就進入監督系統了，在這種情形時，您將在螢光幕上看到充滿了亂七八糟的字，只有在螢光幕底端的左下角有一個星號（＊），星號的旁邊就是一閃一閃的游標。這個星號就是監督系統的標示字。

如果您的APPLE Ⅱ具有自動啓動監督系統（不論是APPLE Ⅱ PLUS上加入的功能或是標準的設備），您必須使用整數BASIC或是APPLESOFT 處理監督系統；當您看到BASIC的系統標示（>表示整數BASIC，］表示APPLESOFT）之後，打入下面的命令:

```
CALL -151
```

這個敍述實際上就是呼叫位於十六進位 FF69位置的一個副程式，但是由於BASIC不認識十六進位數字，所以您必須在這個命令中使用相等的十進位數字代替。只要您輸入這個敍述之後，星號及游標就會出現，這時您就進入監督系統了。

離開監督系統

這裏有幾種方法使您能夠離開監督系統而返回BASIC，它們與您想要離開時所能做的事有極大的關係。如果您想將BASIC程式內的敍述及變數保存下來，您就在打入CTRL -C之後按下 RETURN 鍵離開監督系統；在您按下 RETURN 鍵後，您就會看到BASIC語言的系統標示，這時候，您就可以列出變數的值及程式列（如果這時記憶體內有

程式的話）。

爲了說明這種情形，讓我們假設您使用POKE 敍述將一個值放在一個記憶體位置內，而您又想證明POKE 敍述的功能是否有效；當然您可以使用PEEK 敍述做這個工作，但是監督系統較PEEK及POKE給您更大的能力處理記憶體位置內的資料。下面這個整數BASIC的例子告訴您如何使用 CTRL -C保存BASIC程式及變數（在離開監督系統時）：

```
>10 A=123
>19 REM MOVE CURSOR TO 13TH COLUMN
>20 POKE 36,12
>30 PRINT A
>40 END
>RUN
            123

>CALL -151

   * ← 按 CTRL-C ， 再按 RETURN.

>PRINT A
123
>LIST
   10 A=123
   19 REM  MOVE CURSOR TO 13TH COLUMN
   20 POKE 36,12
   30 PRINT A
   40 END
>
```

如果您想要離開監督系統同時並把目前存在記憶體內的BASIC程式及變數消除，那麼您只要按下CTRL -B及RETURN 鍵就可以了；這樣的輸入會使您回到BASIC語言內，但是在記憶體內的所有程式及變數都會被清除掉。試著將上面例子中的CTRL- C用CTRL- B代替，在由監督系統返回BASIC系統後，您將發現變數A內沒有任何值，而且您也無法列出這個程式。

CTRL -C可以在整數BASIC或硬體的APPLESOFT 中使用，但是無法在由磁帶或磁片輸入的APPLESOFT 中操作。

如果您的APPLESOFT是由磁帶或磁片輸入的，那麼您可以使用CTRL -B離開監督系統，在這種情形時，您將會回到整數BASIC

系統內，而所有您的APPLESOFT 程式及變數都將被清除。下面有一種方法使您回到APPLESOFT 內而不致損壞您的程式及變數。

由監督系統返回到由磁片輸入的APPLESOFT 內，您打入下面的命令：

*3D0G

由監督系統返回由磁帶輸入的APPLESOFT 內，打入下面的命令：

*0G

3DØG及ØG是監督系統內的轉移指令（與BASIC中的GOTO指令同義）。接下去會有更多的介紹。

需要特別注意的是：只能在返回磁帶輸入的APPLESOFT 系統時使用 ØG 命令。

監督系統的功能

監督系統可以從事一些有限的工作，但却是非常有效的；您可以檢查記憶體位置或是微處理機的暫存器，將記憶體內的資料顯示出來或是列印於列表機上，同時也可以改變記憶體及暫存器內的資料。其他的功能包括移動資料區的位置及比較不同資料區內的資料，還有一些有效的功能都將在本章中陸續出現。

檢查記憶體.

下面有三種方法從監督系統內觀察記憶體內的資料：單一位置

（SINGLE-ADDRESS）、字（WORD）及區（BLOCKS）三種方式。單一位置指的是一個數元組的記憶體位置。字就是八個數元組的記憶體位置，可由被八整除的位置開始。至於區則表示一個區域內的記憶體位置，由一個記憶體位置開始而在另一個較高的記憶體位置結束。

檢查單一位置

當您見到監督系統標示時，而您想要觀看單一位置的記憶體，您只要打入這個位置的十六進位數字，後面跟著 RETURN ，就像下面的例子一般。：

```
*FF69
```

監督系統就會將這個位置內的資料顯示出來：

```
FF69- A9
*
```

當您輸入一個位置之後，監督系統就使用它作爲一個指標；因此，如果您輸入FF 69（或者其他任何的十六進位的位置），監督系統就會把這個位置記起來，做爲以後檢查記憶體的起始位置，直到您將它改變後爲止。改變的方法只要打入一個新的位置就可以了。例如，打入：

```
*300F
```

那麼監督系統除了將 3ØØF 位置內的資料顯示出來之外，還把指標改爲 3ØØF 。

檢查"字"的記憶體

假設您在檢查過了 FF 69 的位置之後，想要看看更高位置內的內容，那麼您只要按 RETURN 鍵，監督系統就會繼續的做檢查工作，就像

下面一般：

```
*FF69

FF69- A9
*←——— 按 RETURN 鍵
 AA 85 33 20 67 FD
*

FF70- 20 C7 FF 20 A7 FF 84 34
*←———再按 RETURN 鍵
```

第一次您按下 RETURN 鍵，就會有六個位置的記憶體內容顯示出來，就是從 FF 6A 到 FF 6F 內的資料，在您第二次按RETURN鍵之後，就會有八個數元組的資料顯示，但是最先顯示的是這八個數元組資料的起始位置（ FF 7∅ ）；每一個字（WORD ）的起始位置都能被八整除，這也就是當您第一次按 RETURN 鍵時為什麼沒有顯示位置的原因了，第一次的 RETURN 只是把前面單一位置顯現後剩下包含在同一字內的位置顯示出來而已。

檢查記憶區

您可以在監督系統內檢查一大塊區域的記憶體。輸入起始位置，後面跟著一個句點，最後則是結束的記憶體位置，例如：

```
*F800.F83F
```

監督系統就會把這一塊區域記憶位置內的資料顯示出來：

```
F800- 4A 08 20 47 F8 28 A9 0F
F808- 90 02 69 E0 85 2E B1 26
F810- 45 30 25 2E 51 26 91 26
F818- 60 20 00 F8 C4 2C B0 11
F820- C8 20 0E F8 90 F6 69 01
F828- 48 20 00 F8 68 C5 2D 90
F830- F5 60 A0 2F D0 02 A0 27
F838- 84 2D A0 27 A9 00 85 30
*
```

起始的位置必須等於或大於結束的位置，使您能夠看到較多位置內的資料，如果結束的位置較開始位置爲小，那麼只有起始位置的內容被顯示出來。

您也可以指出一大片的區域，即使超過了螢幕所能顯現的範圍也可以，在這種情形時，資料將往上移動空出空間讓後面的資料顯現，您只能使用 RESET 將這種顯現的情形取消掉，如果您的APPLE Ⅱ擁有自動啓動監督系統，那麼您可以使用 CTRL -S暫時停止顯示的情形，給您一個檢查的機會——但是 CTRL -S並不能使其他的輸出設備暫時停止（例如，列表機）——您可以按空白鍵使顯示繼續。

這種資料區式的檢查方法就稱爲當出資料（DUMP），只要這個工作完成後，指標就會被修正爲下一個記憶體的位置，供您作繼續檢查的工作。

這種檢查資料區命令有一個簡便方法，就是使用指標做爲這個資料區的起始位置，您只需輸入一個句點後面跟著結束的位置就可以了。例如，如果您已經像上面的例子一般地看過了F800到F83F位置的內容，那麼您就可以輸入下面的命令，繼續檢查由F840到F880 的資料區:

```
*.F880
```

這個命令將造成下面的輸出：

```
F840- 20 28 F8 88 10 F6 60 48
F848- 4A 29 03 09 04 85 27 68
F850- 29 18 90 02 69 7F 85 26
F858- 0A 0A 05 26 85 26 60 A5
F860- 30 18 69 03 29 0F 85 30
F868- 0A 0A 0A 0A 05 30 85 30
F870- 60 4A 08 20 47 F8 B1 26
F878- 28 90 04 4A 4A 4A 4A 29
F880- 0F
*
```

檢查微處理機暫存器的內容

在某些時候您也許想要檢查微處理機內暫存器（REGISTERS）的內容。這個工作您可以打入 CTRL－E再按下 RETURN 鍵即可，它的結果應該與下面的相似：

```
A=CD X=B1 Y=C3 P=B5 S=F0
```

這些被顯示的值分別被存在計算器（ACCUMULATOR(A)）、脚註暫存器X(INDEX REGISTER X(X))、脚註暫存器Y(INDEX REGISTER Y(Y))、處理機狀況暫存器(PROCE-SSOR STATUS REGISTER(P))及先入後出指標(STA-CK POINTER(S))內；這些直接被放在等號右邊的值就是這些暫存器內的值；然而，它們並不能借操作監督系統而改變。換句話說，也就是這些暫存器的值是由監督系統存入的，而且直到您執行您的組合語言程式或返回到BASIC中時，才會被改變。

改變記憶體的內容

改變記憶體的內容較檢查它複雜得多，您必須指出那一個位置是要被改變，同時還要把所要放入這個位置內的新資料供應給監督系統。您可以使用單一位置（一次一個數元組）的方式改變記憶體，也可以改變一連串的記憶體位置。

改變單一位置的記憶體

改變單一位置記憶體的第一步就是先設定監督系統的指標，這個指標與前面檢查記憶體時所用的指標相同，由於這兩種的命令都使用同樣

的指標，您可以如同設定檢查記憶體時的指標一般地設定改變記憶體的指標。打入您想要改變的記憶體的十六進位位置，然後按下RETURN鍵。

*1200

就把位置的指標定在十六進位 12ØØ 的記憶體上了。

這時監督系統就會把這個記憶體位置的內容顯現出來：

1200- 73

注意的是，這個輸入造成與單一位置檢查命令產生的結果相同。監督系統的反應顯示出指標所指的位置（ 12ØØ ），同時也將這個位置內的資料顯示出來（在這個例子中就是 73 ）。

下一步就是改變這個位置內的值了；首先您打入一個冒號（ ： ）後面接著您要輸入的新資料的十六進位數字。例如：

*:5F

冒號代表監督系統內改變記憶體內容的命令， 5F 表示新的資料，您可以使用一個命令而不必如上面般的使用兩個命令改變記憶體。如下面一般：

*1200:5F

這個命令行與前面分開兩行的命令具有改變記憶體位置 12ØØ 內資料的同樣功能，在您執行這個命令時，位置指標被修正爲 12ØØ ，而監督系統將 5F 放入這個位置內。這時位置指標內放的就是下一個較高的記憶體位置了。所以如果您想要改變 12Ø1 位置內的資料爲 7F ，您只需打入：

*:7F

而監督系統就會自動地將這個位置的內容改爲 7F 。這時指標就會自動

地加1，而您也就可以如上面一般地改變12Ø2 位置的內容了。

改變更多的記憶體內容

監督系統允許您一次改變不只一個位置的記憶體內容，但是唯一的條件就是它們必須是連續的記憶體位置（例如由12ØØ 到12Ø7 的位置）。這個命令與單一位置的改變命令採同樣的開始方式；首先您先設定位置指標，接著使用冒號告訴監督系統這是一個改變記憶體內容的命令，最後，輸入所有您想要改變的資料，每一個十六進位數字代表的資料必須用一個空白分開。

例如，將ØØ到Ø7的資料放在由12ØØ 到12Ø7 連續的記憶體位置內，您輸入：

```
*1200:00 01 02 03 04 05 06 07
```

事實上您可以改變不只八個的記憶體位置的內容，如果您在這個命令的起始先設定指標，那麼您可以最多在一個命令行內輸入83個不同的值；如果您不需要設定指標值，那麼您可以輸入84個不同的值，在這兩種情形時，這個命令行都會佔據幾個顯示螢幕行的位置，這在實際使用上並不方便，因爲這使得檢查的工作變得困難得多。更讓您爲難的是，在修正錯誤的時候只有使用回返鍵回到那個地方，然後在修正完畢後再重新打入後面的資料。但是如果您堅持在同一個命令行內輸入這麼多的資料，監督系統也能容許您這麼做的。

檢查改變後的記憶體內容

如果您想要最後的結果是正確的，那麼您最好先檢查改變後的記憶體內容，您可以使用前面描述過的三種檢查記憶體的方法達成這個目的。在開始檢查的工作之前，您仍然必須先將位置的指標設定爲最先被改

變的位置。

假設您在前面的例子中已經把 ØØ 到 Ø7 的資料放在由 12ØØ到 12Ø7 的記憶體位置內了，首先您先將指標設定為 12ØØ：

```
*1200
```

監督系統將回應您：

```
1200- 00
*
```

在您再按下 RETURN 鍵之後，您可以看到您所改變的八個位置中的另外七個內所存放的資料：

```
* ←—— 按 RETURN
 01 02 03 04 05 06 07
*
```

如果您所改變的不只是這幾個位置的內容，那麼您可以繼續按 RETURN 鍵，直到您看到您所改變的最後一個位置內的資料時為止。

您也可以使用資料區的方式檢查區域內的資料：

```
*1200.1207
```

這時監督系統給您的回應就是：

```
1200- 00 01 02 03 04 05 06 07
*
```

修正錯誤

如果在您所改變的位置內有錯誤的資料，那麼您可以單獨地修正而不必重新輸入所有的資料，最簡單的方法就是您輸入這個位置然後依照前面所描述的修正單一位置的方法輸入正確資料就可以了。

例如，您在將 ØØ 到 Ø7 的資料輸入到 12ØØ 到 12Ø7的連續位置內時，在 1204 位置處發生了錯誤，那麼您只需要輸入：

```
*1204:04
```

就可以把這個位置的錯誤修正了。下一步驟就是您再回頭檢查一下您剛才所做的是否正確，然後再檢查其他部份的資料確定一切都是正確的。

改變微處理機暫存器的內容

改變微處理機暫存器的過程與改變記憶體的過程稍有不同，因爲暫存器並不如記憶體般地用位置表示。想要改變暫存器的內容，首先您必須使用 CTRL -E 命令檢查它們的值，後面緊跟著輸入檢查暫存器的命令，然後您就可以使用一個冒號（通知監督系統這是一個改變內容的動作），後面跟著由 1 到 5 個的十六進位的值，其中用空白做爲分界。

第一個十六進位的數字將是計算器的新值，第二個值則代替腳註暫存器X內的值，第三個值則是腳註暫存器Y的值，第四個數字變成處理機暫存器的值，至於第五個值就變成先入後出指標內的值。您在改變某一個暫存器內的值的時候，您必須先輸入在這個暫存器之前所有暫存器的值，並且把這個所要被改變的暫存器的新值放在最後。

現在我們舉一個例子，假設您想要改變腳註暫存器Y內的值，而不變動其他暫存器，那麼首先您使用 CTRL -E 檢查這些暫存器。

```
*←—— 按 CTRL-E, 再按 RETURN
A=CD X=B1 Y=C3 P=B5 S=F0
*
```

（現在您所看到的暫存器內的值只是一個例子而已）。

想要改變腳註暫存器Y內的值爲十六進位的 8A 而不改變其他的暫存器，您必須打入計算器內及腳註暫存器X內的值，後面跟著腳註暫存器Y內的新值。

```
*: CD 81 8A
```

至於想要改變除了計算器A外其他暫存器內的值，您必須重新輸入在這個暫存器前面所有暫存器的值，後面跟著這個要被改變的暫存器的新值。

您必須使用 CTRL-E命令然後按 RETURN 鍵檢查暫存器，否則監督系統將認爲您想要改變一個記憶體位置；在檢查暫存器的時候，就通知了監督系統由改變一個記憶體位置而轉換成改變暫存器本身了。

我們現在再對暫存器的改變做一個說明，假設您想改變先入後出指標（當在檢查暫存器時，它的值跟在S暫存器的後面）內的值爲4B，那麼首先您檢查暫存器：

```
* ←—— 按CTRL-E,再按 RETURN
 A=FF X=CD Y=81 P=8A S=FO
*
```

現在依照順序輸入其他暫存器的現值，然後輸入最後一個的新值：

```
*:FF CD 81 8A 4B
```

您仍然要用CTRL -E檢查暫存器內的值是否是如您所想要的。

APPLE II 週邊設備與記憶體的交通

監督系統允許您使用一個磁帶機將一塊區域記憶體位置內的資料存在磁帶上；如果您建立了一個高解析度的造型表（參看第六章），或者您設計了一個組合語言程式，而您又想把它們儲存起來，那麼您可能就需要使用磁帶來保存它們。如果您已經將DOS存在記憶體內了，那麼您可以更快、更準確地使用磁片儲存更多的記憶體資料。如果您想要將記憶體內的資料存入磁片內，您必須暫時地離開監督系統，然後使用BASIC中的DOS命令儲存或取用記憶體內的資料。

將記憶體內的資料存在磁帶內

您必須使用監督系統內寫（即輸出）的命令將記憶體內的資料存在磁帶上；在存的時候，您同時要指出記憶體的起始及終結位置。將記憶體內容存入磁帶內的命令由您所要儲存的記憶體區域的起始位置開始，後面跟著一個句點，然後就是這塊區域的終止位置，最後則是一個W字。

例如，下面這個命令：

```
*2200.2FFFW
```

通知監督系統將由十六進位 2ØØØ 開始到 2 FFF 爲止的記憶體位置內的資料存在磁帶內。

將記憶體內資料寫出的命令並不能檢查它所送出的資料，同時它也無法檢查是否有一個正在動作的磁帶機與主機連接在一起，所以在您做儲存的工作時，您必須確定磁帶機的一切都是正常的。

當您輸入將記憶體內資料寫出的命令時，除非您確定您的磁帶機已經在**RECORD**情況下，並且磁帶已經在磁帶機內正常的轉動了，您才可以按下 **R**ETURN 鍵。如果您的磁帶是處於開頭的地方，那麼在您按下 **R**ETURN 鍵之前，您最好先讓磁帶機轉動五秒鐘左右，讓磁帶前面一段沒有磁性的部份通過。

當您按下 **R**ETURN 鍵之後，電腦將等待10秒鐘之後才會輸送資料，這麼做的原因是使磁帶機能夠將前面的一些訊息（音樂、聲音或是一些機器能瞭解的資料）消除掉，這時電腦會送出一個處理的音調給磁帶機，當您往後使用讀資料的命令時，監督系統將使用這個音調做爲鎖住記憶體的訊號（這將在下一節討論）。

當這個寫出資料的命令完成後，**APPLE** Ⅱ的發聲器將發出"嗶"的一聲，然後您就可以看到監督系統的系統標示了。

寫出記憶體內資料的命令允許您寫出由 1 個數元組到 64**K**數元組的

資料到磁帶上；監督系統輸送出資料的速度大約是每秒 210 個字，在將最後一個數元組的資料送出之後，監督系統會送出一個存有輸出數元組總數的數元組到磁帶上，當您使用讀回資料的命令時，這個數元組就被使用來檢查輸入的資料是否正確。

由磁帶內取出資料

這個讀的命令使您能夠由磁帶機將資料讀回記憶體內；執行這個命令時您必須輸入資料讀入後存放在記憶體的起始位置，後面跟著一個句點，再跟著記憶體的結束位置，最後則是R字。

例如，下面這個命令：

```
*2000.20FFR
```

表示將資料由磁帶上讀入記憶體內，放置的位置由十六進位的 2000 開始到 20FF 為止。

讀取資料的命令與寫出的命令不同，它將強迫監督系統等待著不做任何動作，直到您按下磁帶機上的PLAY 鍵為止；電腦在作業的時候將等待由磁帶機傳來的通知音調，而在遇到這個通知音調之前，監督系統將把電腦鎖住。所以在您按下PLAY 鍵之前，您必須確定磁帶正位於這個音調開始的地方；您可以仔細地聽，分辨出這個處理音調與實際資料的差別，將耳機上的連接器拔掉，您就可以聽到由磁帶機發聲器發出的磁帶聲音了，這個處理音調是穩定而且中等高低的聲音，而實際資料所發出的聲音則是雜亂的。

您在使用讀取的命令之前，必須先將磁帶機的聲音調整到適當的程度，調整聲音的方法在第二章中我們做過詳細地說明。

讀回的命令期望能讀回如您寫出時相同數目的資料；如果您將1024個數元組的資料存入磁帶內，而您現在只想讀回前面的 256 個數元組，那麼監督系統將依您的願望輸送資料，但是您可能會得到一個錯誤的訊

息，如果您想要讀取比您存入時還要多的資料，同樣的情形也會發生。

在讀回資料時的錯誤情形

監督系統在接收由磁帶輸出的資料之前，最少會花3.5秒的時間傾聽磁帶機發出的聲音，這個過程使得監督系統將處理音調的頻率鎖住，那麼如果磁帶內存有的處理音調少於3.5秒，監督系統將在傳送資料時失掉前面的一部份，而導致檢查總數（CHECK SUM）的錯誤，而且您也無法確定監督系統由磁帶的什麼地方開始讀取資料，因爲監督系統只是將資料由磁帶移入記憶體內而不理會它是否正確；在這種情形時，您就會得到監督系統的錯誤訊息（APPLE Ⅱ發聲器發出"嗶"的聲音，而後您將看到ERR的訊息及監督系統的系統標示）。爲了修正這個錯誤，您必須將磁帶轉回到起始的地方，然後將耳機與APPLE Ⅱ連接的接頭拔下，按下PLAY鍵並計算發出音調的時間，如果在您聽到傳送資料之前的聲音少於3.5秒，那麼您必須重新將記憶體內的資料寫入磁帶內。這種情形發生的可能原因是也許您忘了在記錄資料之前先將磁帶轉過不具磁性的部份。

不論您想要讀回較原先存入時更多或更少的資料都可能會使您在APPLE Ⅱ的螢光幕上看到錯誤訊息。在寫出命令中最後一個被送到磁帶機內的數元組就是檢查總數的數元組，它的值完全按照有多少的資料被寫入磁帶內而定；當一個讀取的工作執行完畢後，發現讀入資料的長度與原先寫入資料的長度不同時，監督系統就無法預測那一個資料是檢查總數的數元組，這時監督系統將假設它由磁帶讀入記憶體內的最後一個數元組後面跟著一個檢查總數的數元組。當執行讀回資料命令時，監督系統本身將對讀入的資料做一個檢查總數的工作，然後與自磁帶內讀回來的檢查總數數元組做比較，如果這兩個數元組並不符合，那麼監督系統就會顯示一個錯誤訊息。當然也可能有機會碰巧兩個檢查總數的數元組符合；因此，一般的情況下，您應該讀回如您存入磁帶時同樣數

目的資料，這樣才能使監督系統替您做有意義的檢錯工作。在這一章稍後的地方您將看到如何使用監督系統，證明讀回資料的動作是否正確。

將記憶體資料存入磁片內

使用**APPLE Ⅱ**的**DOS**，您可以更快、更安全地將資料存入磁片內，它的規則與將記憶體資料存入磁帶內的規則稍有些不同，因爲您在將記憶體資料存入磁片內時，您使用的是**BASIC**語言而不是監督系統。雖然如此，這仍然是一個較操作磁帶作業的監督系統命令爲佳的機器階層的命令，在您閱讀這一節之前，您必須對**DOS**有十分的瞭解（參看第五章），而在您開始作業之前，**DOS**必須在記憶體內（參看第二章）。

如果您現在正處於監督系統內，您必須先暫時地進入**BASIC**語言（**DOS**也存在）內，如果您擁有自動啓動監督系統，那麼您就可以使用 **CTRL -B**或**CTRL -C**達成這個目的，如果您擁有的是標準的監督系統，那麼您就必須輸入 **3DØG**命令了。

下面就是一個**DOS**命令的例子，這個命令將記憶體內的資料存入磁碟內：

```
BSAVE SHPTABLE, A$3000, L256, S6, D1, V201
```

這個命令將在磁片上建立一個叫做**SHPTABLE**的**DOS**檔；第二個元素**A$ 3ØØØ**將把由十六進位 **3ØØØ** 開始的記憶體位置內的資料存到磁片上，**A**就表示位置；第三個元素，**L**256 指出這個寫出記憶體命令所要寫出記憶體的長度（**L**），這個例子中就是十進位的 256，**L**的最大長度是十進位的 32767（十六進位的 **$7FFF**）；**BSAVE** 命令允許您在位置（**A**）及長度（**L**）元素內使用十六進位或是十進位的數字，如果您使用的是十六進位數字，那麼您必須在這個數字之前加上一個錢號（**$**）。

在這個例子中的最後的三個元素（S6，D1，V201）都是可以被省略的，它們指出您所要使用的是那一個磁碟；如果您有不只一個的磁碟控制卡插在主機上，而且您想要把資料存在某一個磁片上，而存有這個磁片的磁碟機號碼並沒有在前面被設定，這時您才需要使用S（擴充接點）這個元素；D（磁碟號碼）元素是非常有用的（但並非必須的，只有在您所要使用的磁碟機並不是前面被設定的磁碟機時，您才需要使用這個元素。V（磁碟片號碼）是您所要將記憶體內資料存入某個磁片內時，該磁片所擁有的磁碟片號碼，如果您在BSAVE敍述中所指出的磁碟片號碼與磁片上的不相同，那麼您就會得到一個錯誤訊息。

由磁片內取回記憶體資料

就如您使用BSAVE將記憶體資料存入磁碟內一般，BLOAD 命令將資料由磁片上取回並存入記憶體內。

下面就是一個BLOAD的例子：

```
BLOAD SHPTABLE, A$3000
```

這個命令將現在被設定的磁碟內叫做SHPTABLE 的檔存入記憶體中，存入的起始位置由十六進位的3ØØØ 開始；這個命令也能接受十進位的位置表示。如果這個檔存入記憶體的起始位置與它被存入磁片時所在記憶體內的位置相同時，A（起始位置）這個元素就不需要了，只有在起始位置與BSAVE 命令中的起始位置不同時，您才需要使用這個元素。長度元素（L）是可以被省略並不需要指出，DOS 會檢查這個檔的長度，而且在讀完之後會自動地結束這個命令。至於那些可以被使用在BSAVE中的元素（S，D及V）也可以被使用在BLOAD命令中。

需要注意的是，不要使用這個命令把資料讀到DOS 所佔用的記憶體位置、APPLESOFT 編譯器佔用的位置、文字資料及圖形頁佔用的位置、或是BASIC變數所佔用的位置。如果您這麼做了，很可能使

這個區域內的資料非常混亂，而且使您失掉您想要保存的資料。

移動及比較記憶體區

如果您使用磁帶或磁片讀取或寫出記憶體資料，監督系統有兩個功能可能對您有些幫助。移動的功能將一塊記憶體內的資料複製到另外一個區域的記憶體內；比較的功能把兩塊記憶體內的資料互相比較並報告出兩者不同的地方。當您使用監督系統的讀及寫的命令時，這兩個功能可以保證您所讀或寫的檔內資料是正確無誤的。

移動記憶體的命令

在將資料由記憶體中的一個位置移到另一個位置時，您必須指出移往目的地的開始位置，原始資料所在的開始位置及結束位置。這個命令的格式就是目的地的開始位置，後面跟著一個小於（ < ）的符號，然後就是原始資料的開始位置，一個句點，接著就是原始資料的結束位置，最後才是一個M字（表示移動）。這個命令如同其他監督系統的命令一般，所有的位置都用十六進位數字表示。

例如，下面這個命令：

```
*1200<2000.2100M
```

將資料移到 12ØØ 位置開始的記憶體內，原始資料的開始位置是 2ØØØ，而結束位置則是 21ØØ 。在這個例子中，監督系統將自 2ØØØ 位置開始到 21ØØ 位置結束的記憶體內的資料複製在由 12ØØ 開始到 13ØØ 結束的記憶體內，由於位置都是用十六進位數字表示的，所以這個移動的長度就是十進位的 257 個數元組，或者說是十六進位的 1Ø1 個數元組；至於原先存在 2ØØØ 到 21ØØ 位置內的資料並不會因爲這個移動命令而改變。

當您在移動的命令中指出結束及開始的位置時，原始資料的開始位置必須小於或等於結束位置，如果結束位置小於開始位置，那麼監督系統將只會把由開始位置算起的一個數元組移往目的地的開始位置，然後就結束這個移動的命令。

塡滿記憶體

移動記憶體內容的命令同樣也可以塡滿（FILLS）記憶體，塡滿的過程就是重複地移動一個或者多個數元組的資料到連續位置的記憶體內。假設您想將由1DØØ開始到1DFF結束的記憶體用零塡滿，那麼您可以使用改變及移動記憶體的命令將這一塊區域內的記憶體資料全部轉換成您想要設定的值。

在開始的時候，您可以將零放在這塊區域的第一個數元組內：

```
*1D00:00
```

這就是塡滿記憶體的第一個步驟，第二步驟則使用移動記憶體的命令將一個或多個數元組內的資料複製到一連串的記憶體內。

指出一個開始的位置，這個位置必須較前面已經被設定值的位置要大（在這個例子中就是（1DØ1），設定原始資料的開始位置（在這個例子中就是（1DØØ），然後將原始資料的結束位置設定爲您想要塡滿區域的最後位置（1DFF）減去前面已經被塡入資料的長度後得到的差數（在這個例子中就是1DFF）。

然後我們執行下面這個命令：

```
*1D01<1D00.1DFEM
```

它將由1DØ1開始到1DFF結束的記憶體內塡滿了零（或者說，用位置1DØØ內的值將這塊區域塡滿）；這個過程只能在您所想要塡滿的值是被放在這個資料區的起始處時才能發生作用。下面就是發生的過程；由

於移動目標的開始位置只在原始資料位置的後面一個數元組，所以監督系統首先由 1D∅∅ 移動資料到 1D∅1 去，也就是將 ∅∅ 移往 1D∅1，當第二個數元組被移動的時候，監督系統就取出 1D∅1 位置內的資料並把它移到 1D02 的位置內；這個過程就重複地被執行，直到 1DFF 位置內的資料（在前面一個移動過程中被設定為 ∅∅）被移到 1DFF 位置內時為止。您可以檢查由 1D∅∅ 到 1DFF 位置內的資料，您將發現它們確實都變成了零了。

您也許想要使用不只一個數元組的資料填滿記憶體；例如，您想要將 00 5E 7F FF 四個數元組的資料填入由 1D∅∅ 開始到 1DFF 結束的記憶體內，那麼首先您必須改變由 1D∅∅ 開始的四個數元組資料：

```
*1D00:00 5E 7F FF
```

現在這些資料已經準備妥當了，您就可以使用下面的命令，填滿到 1DFF 為止的記憶體了：

```
*1D04<1D00.1DFBM
```

需要注意的是，移動目標的開始位置是在前面設定資料位置後一個數元組，而原始資料的起始位置就是這塊要被填滿資料的起始位置，至於原始資料的結束位置則是這塊將被填滿資料的最後位置減去前面設定資料的長度後得到的差：

```
 1DFF
 -04     （十六進位運算）
 1DFB
```

如果您試了這個例子，那麼您仍然可以使用檢查的方法檢查出由 1D∅∅ 到 1DFF 位置的資料區內都填滿了前面設定的資料。

比較記憶體的命令

比較記憶體的命令將兩塊區域內的記憶體互相比較，並將這兩塊間的差別記錄起來。您可以將這個命令與讀取及寫出記憶體的命令合用；如果您將記憶體內的資料存到一個週邊設備內，而您想要確定寫出的資料是否正確時，您也可以使用這個比較的命令。

這個命令的格式與移動記憶體的命令相類似；首先輸入目標的起始位置（由何處開始比較記憶體），後面緊跟著小於符號（＜），然後是原始資料的起始位置（由何處開始與目標記憶體比較），一個句號，接著就是原始資料的結束位置（比較動作的最後一個比較的數元組），最後的一個字就是V字表示比較的動作。

下面就是一個例子：

```
*32D0<0.CV
```

這個命令通知監督系統開始將32DØ位置內的資料與Ø位置內的資料比較，並且繼續依照順序兩兩相比，直到32DC位置內的資料與ØØØC內的資料比較完畢後才停止。當您在一個監督系統的命令行中時，表示記憶體位置數字前面的零是可以省略的。

如果監督系統在比較的過程中遇到一個原始資料中的數元組與目標資料中相對應的數元組中的資料不相同時，那麼這個數元組的位置及其中包含的資料與目標資料中相對應的數元組內的資料同時會被顯示在螢光幕上。

例如，您將ØØØØ到ØØØC位置內的資料移到32DØ到32DC位置的記憶體內：

```
*32D0<0.CM
```

並且將這兩個區域內的資料顯示出來：

```
*0.C

0000- 4C 3C D4 4C 3A DB 8C 8C
0008- FF FF 4C 99 E1
*32D0.32DC

32D0- 4C 3C D4 4C 3A DB 8C 8C
32D8- FF FF 4C 99 E1
*
```

您可以用眼睛檢查這個移動記憶體的操作是否正確；如果您將32D8位置內的資料由FF改爲5A：

```
*32D8:5A
```

接著您輸入比較記憶體的命令：

```
*32D0<0.CV
```

這時監督系統就會一個數元組接著一個數元組地比較原始資料與目標資料區內的資料，直到比較到ØØØ8位置與32D8位置內的資料時，由於32D8內的資料剛剛才被改正過，所以監督系統就會將下面的比較結果顯示出來：

```
0008-FF (5A)
*
```

也就是說在原始資料ØØØ8位置內的資料與相對應的目標位置內的資料並不相符合。監督系統首先顯示的是原始資料的位置及其中所包含的資料（ØØØ8位置內的資料是FF），接著才將相對應目標位置（32D8）內的資料（5A）包含在括弧內顯示出來。

目標資料區的位置並不會被顯示出來，因爲監督系統假設您可以用十六進位數字爲底的加法輕易地算出來。下面給您一種不必計算的方法，就是將原始資料區及目標資料區放在命令中的位置調換：

```
*0<32D0.32DCV
```

您會得到下面的結果：

```
32D8-5A (FF)
*
```

這個訊息顯示原始位置 32D8 內包含著 5A 的資料，而相對應的位置內（就是前面的 ØØØ8 位置）包含著 FF ；這個方法雖然消去了一個計算的工作，但是您必須重新計算一個新的原始資料的結束位置（3 32DC ）。

比較記憶體與週邊設備內的資料

如果您將記憶體內的資料存到磁帶或磁碟內時，這個比較的命令將會非常有用。當您將一段的記憶體資料存到週邊設備內然後又將它們存入記憶體內不同的位置內時，您可以比較存入到記憶體內的資料是否正確。下面的這個例子對於您將組合語言程式、造型表及其他的一些資料照上面的過程作業，並且作一個說明。

如果您使用磁帶作記錄記憶體內資料的工具，第一步您就必須輸入下面寫出的命令：

```
*2000.20FFW
```

這個命令將由 2ØØØ 開始到 2ØFF 結束的記憶體內的資料寫到磁帶上去。在您按 RETURN 鍵之前別忘了使磁帶機處於RECORD的狀況下。只要APPLE Ⅱ的發聲器送出"嗶"的一聲時，那就表示這個寫的動作已經完成了，這時您就可以停止磁帶的轉動並且將磁帶轉回到處理音調開始的地方。假設由 21ØØ 到 21FF 位置的記憶體都是可供使用的，那麼您可以使用讀取資料的命令將由 21ØØ 開始到 21FF 結束的記憶體內放入磁帶內的資料：

```
*2100.21FFR
```

別忘了按下PLAY 鍵啓動磁帶機；當APPLE Ⅱ發聲器送出一聲"嗶"的聲音時，這個讀取資料的動作就完成了。這時您就可以比較記憶體內與磁帶內的資料是否相同：

```
*2000<2100.21FFV
```

這個比較的命令將自 2∅∅∅ 開始到 2∅FF 結束的記憶體內的資料與曾經被寫出到磁帶上而又被讀回到 21∅∅ 開始的位置內資料相比較，如果沒有任何不相符的地方被顯示出來，那麼您就可以確定這個寫出的命令是成功地作業完成了。

比較記憶體與磁片上的資料，您可以採取與上面相同的過程。在您使用磁碟機讀取或儲存資料之前，DOS 必須先被存在記憶體內；第一步使用B SAVE命令將一塊區域的記憶體存到磁碟內：

```
BSAVE MEMDATA, A$2000,L$FF
```

然後您可以使用BLOAD命令將前面被存到磁片內的MEMDATA檔讀回記憶體內：

```
BLOAD MEMDATA, A$2100
```

注意的是，在這個敍述中的位置元素較前面原本資料所在的位置多256 個數元組（十進位的數字）。當MEMDATA檔被讀入記憶體內後，您就可以使用比較的命令比較這兩塊區域內的資料了。

```
CALL -151
*2000<2100.21FFV
```

如果沒有任何不相符的地方顯現出來，那麼您就可以確定記憶體內的資料已經被正確地存入磁碟內了。

GO命令

監督系統中的一個命令可以將APPLE Ⅱ的控制權轉移到您所指定位置上的一個程式中去。在這一章開始的地方，您曾經看到如何離開監督系統而回到具有磁碟系統的APPLESOFT 語言內。下面這個命令：

```
*3D0G
```

通知監督系統跳到記憶體 3Dø 的位置去，並且執行存在這個位置內的機器語言指令。在這個命令行結尾的G字就表示GO；如果您輸入上面這個命令而且 DOS 已經在記憶體中了，位置 3Dø 內就存有這個組合語言指令的第一部份。

```
JMP  $9DBF
```

當監督系統將控制權轉移到 3Dø 內的指令去時，執行這個指令的結果就使控制轉到 9DBF 的位置去，而這個位置正好就是 DOS 程式開始的地方。

GO 命令的格式就是您想要轉移的位置後面再跟上一個G字；這個位置是可以被省略的，如果您沒有輸入任何位置，那麼監督系統就會使用記憶體的指標做爲這個命令所用的記憶體位置。

使用列表機

如果您的APPLE Ⅱ使用序列輸出介面卡(SERIAL INTE-RFACE)或訊號交換介面卡(COMMUNICATIONS CARD)與列表機連接，作爲輸出的工具，那麼您就必須輸入與介面卡連接的擴充接點號碼，後面跟著CTRL -P及RETURN 才能將輸出由螢光幕

轉移到列表機上。只要您輸入了這個命令，那麼應該輸出到螢光幕上的資料，就會被輸出到列表機去了。如果您想再用螢光幕作爲輸出的工具，那麼您只需將上面命令中擴充接點的號碼用ø代替就可以了。

當您使用這個命令時，您必須確定您所選擇的擴充接點上已經插有介面卡，如果介面卡並沒有被插在擴充接點上，那麼**APPLE Ⅱ**將被鎖住，而唯一的恢復方法就是按 **RESET** 鍵。

這個列表機的命令與**BASIC**語言中的**PR#**（擴充接點的號碼）命令具有相同的功能，它們同樣地在54位置（$36）內設定兩個數元組的**CSW**（**CHARACTER OUTPUT SWITCH**）值，這兩個數元組內存有指向現在被使用的輸出字程式的位置。您使用 **CTRL -P** 改變擴充接點時，您同時改變了**CSW**內的值。

鍵盤命令

這個命令告訴監督系統從某個設備接受輸入的資料，而不是由鍵盤接受資料；就如同列表機命令一般，您指出這個設備所在的擴充接點號碼，後面跟著 **CTRL -K**，再按下 **RETURN** 鍵；當您想要回返到由鍵盤接受資料時，您只需將鍵盤命令中擴充接點的號碼改爲ø就可以了。

這個命令設定在位置56（$38）開始的兩個數元組的**KSW**（**KEYBOARD INPUT SWITCH**）值，這個值就是一個指向擴充接點的位置。

設定顯示的情況

輸入使螢幕顯示爲黑白相反的命令——簡字爲**I**，就可以使監督系統的輸出在螢光幕上呈現黑白相反的情形。這個命令使得顯示在螢幕上的字變成黑色而螢幕的底色爲白色，但是，您輸入的任何監督系統的命令，在螢光幕的顯示上仍然是白字黑底的正常情況。

您可以輸入使螢光幕顯示正常的命令－－簡寫為N，使黑白相反的顯示情形變為正常。這兩個命令除了I或N外不需要其他任何的元素。

在監督系統中做八個數元(BIT)的運算

監督系統可以執行八個數元(BIT)的加減運算，而運算的結果仍然是八個數元的長度。執行加法運算時，先輸入一個十六進位的被加數，後面跟著加號（＋），再接著是十六進位的加數。如果加後的結果較 FF 為大，那麼監督系統將會把最左的那一個數元捨棄，而將較低次位的八個數元的資料顯示出來，就如同下面的例子一般：

```
*7F+8A
=09
```

执行減法時，首先輸入被減數，後面跟著減號（－）及減數，如同加法一般，這兩個數目都必須是十六進位的數字。如果減出來的結果小於 ∅，監督系統顯示的是這個結果的 1 的補數，就如同下面的例子一般:

```
*0A-2D
=DD
```

由使用者自訂的監督系統的命令

您可以在監督系統內輸入 CTRL －Y命令而自己設立一個命令，當 CTRL －Y被輸入時，監督系統會自動地跳到3F8的位置去，而在3F8的位置內您可以存入一個機器語言的轉移指令；如果您在記憶體內的某處存有一個特別的機器語言程式，那麼您就可以使用CTRL －Y命令而跳到3F8內所存的位置開始的程式去。

下面這個例子就是對於使用 CTRL －Y而重新將由磁碟輸入的 APPLESOFT 重新啓動而不必輸入3D∅G命令。

首先，您必須知道機器語言中轉移指令的格式；這個命令需要三個

數元組，第一個數元組內存的是指令碼—— 4C ，後面的兩個數元組內存的是所要轉移到的位置，但是您必須將位置的後一個數元組放在前面。

下面是一個改變記憶內容的命令，它設定一個轉移的指令將控制轉移到3DØ的位置內去：

```
*3F8:4C D0 03
```

現在您試著輸入 CTRL -Y 命令，您發現不必使用3DØG 就可以做同樣的工作了吧！

我們現在舉另外一個例子，您可以知道如何使用 CTRL -Y跳到 MINI -ASSEMBLER 語言內，這個語言我們將在下一節做詳盡的說明。如果您輸入下面的監督系統的命令：

```
*F666G
```

這時您將看到MINI -ASSEMBLER的系統標示：

MINI -ASSEMBLER 系統開始的位置是 F 666 ，您也可以使用存在3F 8 內的一個 JMP命令跳到這個位置上去：

```
!3F8:JMP $F666
03F8-    4C 66 F6     JMP    $F666 ←—The Mini-Assembler
!                                    displays this line
```

雖然MINI -ASSEMBLER 語言我們尚未討論，上面這一行將這個命令的位置設定爲十六進位 3 F 8 ，而它所指出的位址（ $F666 ）就說明了轉移到MINI -ASSEMBLER 程式開始的位置這件事實。您若要返回到監督系統，那麼您只需輸入下面的命令：

```
!$FF69G
*
```

監督系統的標示就會顯示在螢光幕的左下角，在這時候，如果您接下 CTRL -Y，您就看見到MINI -ASSEMBLER 的系統標示又出現了。利用這種使用者自訂的命令將控制轉移到MINI -ASSEMBLER內，可以省掉您許多敲鍵的工作；當您要將這個使用者自訂的命令消除掉時，您只需在3F8位置內放入另外一個轉移的指令就可以了。

MINI-ASSEMBLER語言

如果您擁有標準的APPLE Ⅱ（或者具有整數BASIC語言卡的APPLE Ⅱ PLUS），您就擁有一個在ROM中的程式，使得您使用機器語言程式時省去許多自己翻譯成機器碼的困難。MINI -ASS -EMBLER與整數BASIC一起被存在ROM內，所謂MINI -ASS -EMBLER的意思是說，程式設計師必須使用位置而不是使用易記的標示字做爲組合語言敍述中的運算元（OPERANDS），同時，您所輸入每一行對應的碼立刻自動地被組合成機器語言，這麼做主要的問題就是您無法插入或消除掉指令，而在一個完整的組合語言中就可以使用文字編輯的方式達成這個目的。

MINI -ASSEMBLER 最主要的好處就是您可以使用組合語言的易記指令而將機器指令直接輸入到APPLE Ⅱ內。

本章後面的部份對MINI -ASSEMBLER 做詳盡的描述並且教導您如何使用它；但是在這一章中並不解釋組合語言的程式技巧，也不會對APPLE Ⅱ所使用的65∅2 組合語言指令做全部的說明。

所以如果您對組合語言、運算元及易記符號等都沒有興趣的話，您現在就可以停止閱讀接下去的內容。如果您想繼續，那麼您首先必須先對組合語言程式設計及6502 指令先做一番研究，然後才能夠讀完這一章。

進入MINI-ASSEMBLER

MINI-ASSEMBLER 語言的進入點就是十六進位的F666位置。如果您想由監督系統進入MINI-ASSEMBLER，您輸入下面的命令：

```
*F666G
```

這將使監督系統跳到MINI-ASSEMBLER內。由整數BASIC或是APPLESOFT進入MINI-ASSEMBLER，您必須輸入下面立即式的命令：

```
CALL -2458
```

當您進入MINI-ASSEMBLER時，APPLE Ⅱ的發聲器就會發出一聲"嗶"的聲音，而MINI-ASSEMBLER的系統標示——驚嘆號（!）就會顯示在螢光幕上。

輸入時的錯誤

MINI-ASSEMBLER當您在輸入一個組合語言的指令時會替您做偵錯的工作，它會使發聲器發出"嗶"的一聲，並且將這個指令重新顯示出來，而在第一個不正確的字下面用一個脫字記號（∧）表示錯誤，而這時計算位置的計數器維持原值，直到您輸入正確的指令為止。

在MINI-ASSEMBLER內的監督系統命令

只要您在MINI-ASSEMBLER內，您就可以執行監督系統的命令。在MINI-ASSEMBLER系統標示的後面，只要輸入一個錢

號（ $ ），後面跟著監督系統的命令就行了。下面這個例子告訴您如何由 MINI-ASSEMBLER 內檢查記憶體。

```
!$1CFF
1CFF- E6
```

這個功能替您節省了許多轉換系統的時間。當您在MINI-ASS-EMBLER內時，您可以先輸入一個錢號再輸入監督系統的命令，而執行監督系統的命令；事實上，當您要離開MINI-ASSEMBLER時您必須使用這個功能。

離開MINI-ASSEMBLER 系統

如果您想要離開MINI-ASSEMBLER，您就在錢號後面輸入監督系統命令就可以了；回到監督系統時，必須將控制轉移到FF69的位置去，而這個命令就是 $FF69G。

除非您的APPLESOFT 是由磁片或磁帶輸入的，否則 $CTRL-B 或 $ CTRL -C 會把控制轉移到BASIC 內。至於由磁碟輸入的APPLESOFT，您必須使用 $ 3DØG 命令；由磁帶輸入的APP-LESOFT，則必需使用 $ ØG命令才能回返到BASIC 系統內。

指令的格式

雖然這一節的目的並不是要教您組合語言的程式設計，但是仍然有一些MINI-ASSEMBLER 的概念您必須在使用它之前熟悉它的使用方法。首先，MINI-ASSEMBLER本身具有一個指令指標，這個指標與監督系統內的記憶體指標是不同的，在您輸入指令之前，您必須先設定這個指標的值；第二點就是 6502 微處理機所使用的指令有許

多不同的格式，而這些格式大都依照使用位置的型式而有所不同。

6502微處理機具有八種不同的定位置方式，但是只有六種不同的指令格式；下面就是它們的描述。

第一種就是絕對式的（ABSOLUTE）定位置，這種方式只需要一個或兩個數元組的記憶體位置當做運算元。例如：

```
AND $303A
```

MINI-ASSEMBLER 並不需要在十六進位數字前看到錢號（$），它假設所有表示位置的數字都是十六進位的。

第二種情形就是直接式的定位置方式（IMMEDIATE ADDRESSING MODE），就像下面的例子：

```
LDA #$04
```

注意這個井號，位於運算元的第一個字，我們使用這個字告訴電腦將 Ø4 這個值直接放到計算器內；如果沒有這個井號，那麼 **MINI-ASSEMBLER** 就認爲這個指令是要把記憶體位置 ØØØ4 內的資料取出，然後再放入計算器內，而這種情形就是前面絕對位置的作業方式。

第三種的方法就是腳註的方法（INDEXED METHOD），它的格式就和下面一般：

```
CMP $23,X
```

或是：

```
AND $80,Y
```

這兩個指令都具有X或Y暫存器做爲第二個運算元。基本上，這種格式的方法就是將第一個運算元所表示的位置與第二個運算元X或Y暫存器內的資料相加，然後使用這個和做爲指令所要處理的位置。

接下來就是使用腳註的間接格式（PRE-INDEXED INDIR-ECT FORMAT），就如同下面這個例子：

```
AND ($F0,X)
```

指出位置（$F∅）與暫存器X內的資料相加後得到的和指到在記憶體內前面256個數元組中的一個位置，而這個位置內存有另外一個指向資料的位置，這個資料才是要在這個指令中被用來做爲運算元的實際資料。

至於後訂腳註的間接方式（POST-INDEXED INDIRECT ADDRESSING）就使用下面的格式：

```
ORA ($22),Y
```

這個指令使用第一個運算元做爲指向一個兩個數元組位置的指標，在這例子中就是$22；這個指令將Y暫存器內的值與存在$22位置內的值相加，而所得的和就是這個指令中所使用的資料所在的位置。就如上面的例子一般，MINI-ASSEMBLER分辨指令是否使用後訂腳註的間接格式時，只需要檢驗第二個運算元是否被包含在括弧內就可以了。

間接位置（INDIRECT ADDRESSING）較腳註位置的表示方法來得直接一些。下面就是一個例子。：

```
JMP ($22FE)
```

執行這個指令時，JMP指令並不會把$22FE位置直接存到程式計數器（PROGRAM COUNTER）中，相反地，它會將存在$22FE位置內的兩個數元組的位置存入程式計數器內，因此，在間接位置格式中的運算元只是一個指標而不是一個實際被使用的位置。

使用MINI-ASSEMBLER

我們曾經在前面提過，MINI-ASSEMBLER擁有一個位置計

數器（LOCATION COUNTER），這個計數器內容增加的方式是依照您輸入的組合語言敍述的長度而增加的，換句話說，當您輸入的敍述被組合成機器語言時，MINI－ASSEMBLER就會計算這個機器語言指令的長度（1、2或3個數元組），然後再把這個長度加入位置計數器內而決定下一個程式行的位置。

使用MINI－ASSEMBLER的第一步驟就是設定位置計數器的值，這個工作必須與您輸入的第一個組合語言敍述同時完成。例如：

```
!8DB0:LDA #$04
```

在MINI－ASSEMBLER系統標示的後面直接輸入開始的位置（在這個例子中就數8DBØ），後面跟著冒號（：）及您的第一個組合語言敍述。在您輸入下一個指令之時您就不需要再輸入一個新的位置了，MINI－ASSEMBLER會自動地計算下一個位置，除非您照上面的方式重新設定另外一個位置計數器的值。

只要您設定了位置計數器的值以後，您就可以一次一行地輸入一系列的組合語言指令了。在第一行後面的指令，您必須在輸入指令之前先留一個空白的位置，就像下面一般：

```
! JSR FB1E
```

這就會使MINI－ASSEMBLER自動地去計算下一個位置計數器內的值了。

一個例子

這一節將使用一步一步解釋的方法對MINI－ASSEMBLER的作業方式做一個說明。這個例子的目的是建立一個使用APPLE Ⅱ的遊戲控制器及發聲器發出聲音的程式。這個程式的過程就是使用監督系統內的PREAD副程式（由FB1E位置開始）由控制器Ø及1內讀入

資料，控制器∅輸入的資料就是發聲器發聲時間的長短（∅表示最短，FF 表示最長），而由控制器1輸入的值則是與控制器∅相反的意義（∅表示最長的區域，FF 表示最短的區域）。

這個程式由1D∅∅的位置開始，並且使用1CFF位置儲存由控制器∅輸入的資料。

當您輸入組合語言的程式行時，MINI－ASSEMBLER會將您這個輸入的命令轉換成目前計數器內容、作業碼（OPERATION CODE）及運算元（使用機器語言的格式）而您輸入的指令的程式行就顯現在下一個位置。例如：

```
1D00-    A2 00          LDX    #$00
```

位置計數器的內容顯示在這個行的最前面，後面跟著一個破折號（－），接著是作業碼（A2代表LDX 指令）顯現，後面跟著這個指令的最後一個數元組。在佔用三個數元組的指令時，低次位的數元組較高次位的數元組先顯現；最後就是指令本身。

下面就是這個程式的完整程式列；注意的是，您輸入的每一行組合語言敍述都會被MINI－ASSEMBLER轉變成機器語言而在下一行的位置顯現出來。

```
!1D00:LDX #$00          — 設定位置計數及輸入第一個指令

1D00-    A2 00          LDX    #$00
! JSR FB1E              —所有的數字都是十六進位($不需要加在前面)

1D02-    20 1E FB       JSR    $FB1E
! STY 1CFF

1D05-    8C FF 1C       STY    $1CFF
! INX
1D08-    E8             INX
! JSR FB1E

1D09-    20 1E FB       JSR    $FB1E
! LDA C030

1D0C-    AD 30 C0       LDA    $C030
! DEC 1CFF
```

```
1D0F-    CE FF 1C     DEC   $1CFF
! BNE 1D0C ← Mini-Assembler 計算相關的轉移 (F8)

1D12-    D0 F8        BNE   $1D0C
! LDA C030

1D14-    AD 30 C0     LDA   $C030
! INY

1D17-    C8           INY
! BNE 1D14

1D18-    D0 FA        BNE   $1D14
! JMP 1D00

1D1A-    4C 00 1D     JMP   $1D00
!
```

在您輸入完這個程式後，您必須檢查輸入的工作是否做得正確，最好的方法就是將在記憶體內的程式以組合語言的格式顯現出來，但是您必須如下述般地使用監督系統來做這個工作。

您也許想將這個程式存到磁帶（使用監督系統的W命令）或是磁碟內（使用BASIC語言的BSAVE敍述，而得以安全的保存這個程式。

在執行這個程式時，您必須使用監督系統的G命令或是BASIC語言內的CALL 7427 命令將控制轉移到1Døø的位置去。然後您就可以操縱遊戲控制器使發聲器發出種種的聲音了；若您要結束這個程式的執行，您只需按下 RETURN 鍵就可以了。

將機器語言重組回組合語言的格式

監督系統具有一個命令可以使您把機器語言的指令重組爲組合語言的格式，並且列印或顯示出來，即使您的APPLE Ⅱ的ROM內並不存在MINI-ASSEMBLER系統也可以有同樣的功效。L命令就表示列印（LIST），可以將20個機器語言的指令重組爲組合語言的敍述而顯示在螢光幕或其他您所選擇的輸出設備上。這個LIST命令使用位

置計數器做爲指標，指向下一個要被重新組合的指令位置。因此，如果您在輸入上面這個程式之後就輸入L命令，那麼重組的動作會由1D1D的位置開始，而不會將您輸入的任何程式列印或顯示出來。

所以在使用LIST命令之前您最好能夠先設定位置計數器內的值。下面就是這個發聲程式被重新組合後的程式列：

```
!$1D00L

1D00-   A2 00       LDX   #$00
1D02-   20 1E FB    JSR   $FB1E
1D05-   8C FF 1C    STY   $1CFF
1D08-   E8          INX
1D09-   20 1E FB    JSR   $FB1E
1D0C-   AD 30 C0    LDA   $C030
1D0F-   CE FF 1C    DEC   $1CFF
1D12-   D0 F8       BNE   $1D0C
1D14-   AD 30 C0    LDA   $C030
1D17-   C8          INY
1D18-   D0 FA       BNE   $1D14
1D1A-   4C 00 1D    JMP   $1D00
1D1D-   9F          ???
1D1E-   4E A5 12    LSR   $12A5
1D21-   A4 96       LDY   $96
1D23-   A3          ???
1D24-   D0 A4       BNE   $1CCA
1D26-   EF          ???
1D27-   A4 62       LDY   $62
1D29-   A2 70       LDX   #$70
```

在這個顯示情形中我們發現最後的八個重組後的指令並不具有什麼意義，因爲這個程式在1D1A位置處就已經結束了。

需要注意的是，LIST命令是監督系統提供的一個功能，它與MINI－ASSEMBLER是不相關的（因此在這個命令的前面加了一個錢號）。您輸入L（在MINI－ASSEMBLER中就必須輸入$L），再按下RETURN 鍵，而並沒有設定位置計數器的值，那麼您就告訴監督系統將位置計數器內所存的位置開始的20個指令重組成組合語言的格式而列印或顯示出來。

格式的測試及偵錯

監督系統在MINI－ASSEMBLER語言內提供了標準APPLE Ⅱ機器偵錯的功能，使得您在設計組合語言程式時，爲您提供很大的幫助。設計低層次（LOW－LEVEL）語言的程式最困難的所在往往就是測試及偵錯；並非只是把變數中的值列印出來就能夠完成偵錯的，而是要觀察記憶體位置、暫存器及程式本身您才可以檢查組合語言的程式。STEP及TRACE這兩個監督系統的命令就爲您做這些工作。

STEP及TRACE命令在具有自動啓動監督系統的APPLE Ⅱ中並不能被使用。

STEP命令

您可以藉著執行組合語言程式時發現的各種情況而輕易地觀察出錯誤的情形，但這並不是一個有效的找出錯誤的方法；如果這個程式很小，那麼您可以一步步地執行文，並且檢查每一個指令執行後的結果。STEP命令就是提供您一個具有這樣功能的工具。

當您執行STEP 命令之後，監督系統會把由位置計數器內的值所指的位置中的指令重組成組合語言的格式並把它顯示出來，而且執行這個指令，然後將微處理機的暫存器內的值顯示出來，再將APPLE Ⅱ的控制轉移回到監督系統。

STEP 命令的格式就是一個可有可無的位置（用來設定位置計數器的值），後面跟著S字。

下面這個STEP 命令執行發聲程式的第一個指令，X暫存器內的值是Ø，而其他三個暫存器的值都沒有被改變。

```
*1D00S

1D00-    A2 00        LDX    #$00
 A=FF X=00 Y=8C P=32 S=F8
*
```

您也可以在STEP 命令與其他的監督系統命令的使用中調換它們。我們現在做一個示範，首先您使用步步檢查的方式檢查前面發擊的程式直到位於1D∅5位置的STY指令爲止；在這個過程中您必須輸入九次的S命令，這是因爲在1D∅2位置的JSR指令呼叫一個位於FB1E位置的副程式，而在您返回到1D∅5位置時，您就可以使用監督系統中的檢查記憶體的命令來檢查1CFF位置的內容。在1D∅5位置內的指令把由控制器∅所讀入的值存到1CFF位置內去。

```
*S

1D05-    8C FF 1C     STY    $1CFF
 A=00 X=00 Y=00 P=32 S=F8
*1CFF

1CFF- 00
*
```

如同您在檢查記憶體的命令執行後所見到的，Y暫存器內的值(∅∅)已經被存在1CFF位置內了。您可以在使用STEP 命令時，與其他的監督系統命令合用，增強它的功能。

TRACE命令

有時候您想瞭解一個較長程式的執行狀況，但是由於它的長度，使得您無法一步步地執行每一個指令，這時您就必須觀察每一個指令的執行，但是只在必要的時候才打斷這個程式；而監督系統中的TRACE命令就是爲您做這個工作，它的輸出與STEP 命令的相似，只是它不必如STEP命令般地每檢查一個指令都需要輸入一次 STEP 命令；如果您想要停止TRACE 命令，您可以按下RESET 鍵或是將組合語言

中的 BRK 指令放在您的程式中，當監督系統碰到這個指令時，它就會把電腦的控制權轉回給您。

TRACE命令的格式就是一個可有可無的位置後面跟著T字。下面就是上面那個發聲程式追蹤執行的第一部份：

```
*1D00T

1D00-    A2 00        LDX    #$00
  A=FF X=00 Y=8C P=32 S=F6
1D02-    20 1E FB     JSR    $FB1E
  A=FF X=00 Y=8C P=32 S=F6
FB1E-    AD 70 C0     LDA    $C070
  A=00 X=00 Y=8C P=32 S=F4
FB21-    A0 00        LDY    #$00
  A=00 X=00 Y=00 P=32 S=F4
FB23-    EA           NOP
  A=00 X=00 Y=00 P=32 S=F4
FB24-    EA           NOP
  A=00 X=00 Y=00 P=32 S=F4
FB25-    BD 64 C0     LDA    $C064,X
  A=00 X=00 Y=00 P=32 S=F4
FB28-    10 04        BPL    $FB2E
  A=00 X=00 Y=00 P=32 S=F4
FB2E-    60           RTS
  A=00 X=00 Y=00 P=32 S=F4
1D05-    8C FF 1C     STY    $1CFF
  A=00 X=00 Y=00 P=32 S=F6
1D08-    E8           INX
  A=00 X=01 Y=00 P=30 S=F6
1D09-    20 1E FB     JSR    $FB1E
  A=00 X=01 Y=00 P=30 S=F6
FB1E-    AD 70 C0     LDA    $C070
  A=27 X=01 Y=00 P=30 S=F4
FB21-    A0 00        LDY    #$00
  A=27 X=01 Y=00 P=32 S=F4
FB23-    EA           NOP
  A=27 X=01 Y=00 P=32 S=F4
FB24-    EA           NOP
  A=27 X=01 Y=00 P=32 S=F4
FB25-    BD 64 C0     LDA    $C064,X
  A=27 X=01 Y=00 P=30 S=F4
FB28-    10 04        BPL    $FB2E
  A=27 X=01 Y=00 P=30 S=F4
FB2E-    60           RTS
```

使用TRACE命令較STEP 命令較不方便的地方就是您對程式中每一個步驟的執行情形具有較少的控制權。當您輸入程式的時候，您必須在重要的地方放入 BRK 指令，而這些對於程式的邏輯就造成了障礙

；當然，您也可以使用**NOP**（**NO OPERATION**碼）指令代替**BRK**指令，但是在您結束這個程式的偵錯工作時，可能會使您忽略掉這些額外的內容。

另外附帶說明的一點是**TRACE** 命令在執行的時候會使您的組合語言程式速度變得緩慢得多。例如，假設您將前面發聲的程式中 1D1A 位置處用**BRK**指令代替，然後您輸入 1DØØG 命令執行這個程式，這個程式執行的時間大約不到一秒，但是，如果您輸入 1DØØ**T** 命令，那麼這個程式大約要花費 60 到 70 秒的時間才能執行完畢。因此，如果您想要測試一個較大的程式，又想用**TRACE** 命令來幫助您，您最好謹慎地使用它，以免浪費太多的執行時間。

更進一步瞭解位置計數器

就如同前面所提過的，您可以使用監督系統中大部份的命令與**STEP**及**TRACE**命令交替著使用，但是仍然有一些例外。**LIST**，**GO**及使用者自訂的**CTRL -Y**命令當它們被輸入後，都會改變位置計數器內的值，這就會使您一步步檢查的程式的過程整個被攪亂了，除非您在執行完畢這些指令後立即重新設定位置計數器內的值，否則您檢查的工作就無法繼續下去。下面這個例子就是告訴您位置計數器內的值被這些指令攪亂的情形：

```
*1D00S
 1D00-    A2 00         LDX    #$00
  A=62 X=00 Y=00 P=32 S=F3
 *S
 1D02-    20 1E FB      JSR    $FB1E
  A=62 X=00 Y=00 P=32 S=F3
 *L                                      ← 列出命令
 FB1E-    AD 70 C0      LDA    $C070
 FB21-    A0 00         LDY    #$00
 FB23-    EA            NOP
 FB24-    EA            NOP
 FB25-    BD 64 C0      LDA    $C064,X
```

```
FB28-    10 04        BPL    $FB2E
FB2A-    C8           INY
FB2B-    D0 F8        BNE    $FB25
FB2D-    88           DEY
FB2E-    60           RTS
FB2F-    A9 00        LDA    #$00
FB31-    85 48        STA    $48
FB33-    AD 56 C0     LDA    $C056
FB36-    AD 54 C0     LDA    $C054
FB39-    AD 51 C0     LDA    $C051
FB3C-    A9 00        LDA    #$00
FB3E-    F0 0B        BEQ    $FB4B
FB40-    AD 50 C0     LDA    $C050
FB43-    AD 53 C0     LDA    $C053
FB46-    20 36 F8     JSR    $F836
*S                                     — 改變位置計數器，所以
                                         下一步是 $FB49
FB49-    A9 14        LDA    #$14        而不是 $FB1E
 A=14 X=00 Y=00 P=30 S=F6
*S

FB4B-    85 22        STA    $22
 A=14 X=00 Y=00 P=30 S=F6
*
```

有用的監督系統副程式

在某些情況下，BASIC語言系統無法滿足您所要求的功能，而這就是爲何程式設計師希望由BASIC語言內使用到組合語言副程式的原因了；在這一節中我們將教您如何在BASIC程式中使用這些副程式。

您可以將組合語言的程式加入BASIC程式中，藉以解決許多問題。但是，您準備把這些組合語言程式放在記憶體的什麼位置呢？記得APPLE Ⅱ的記憶體擁有四個保留區域（文字資料／低解析度圖形頁及兩個高解析度圖形頁），而且DOS及APPLESOFT的編譯程式也會佔用記憶體的位置。所以將一個程式放在不致影響其他程式的位置的問題就完全依賴記憶體的大小及APPLE Ⅱ的型式而決定了。

監督系統具有三個最好的原因供您選擇做爲組合語言的副程式：第一，它被存在ROM內，您不必擔憂應該將它存在什麼位置內；第二，

監督系統的程式都已經被偵錯無誤後才被放入ROM內的；最後就是它不必佔用任何記憶體的位置。這些有用的監督系統的副程式我們在附錄D中會有詳盡的說明。

與這些副程式合併使用

如果您決定在BASIC程式內使用一個監督系統的副程式，那麼首先您要確定的是沒有BASIC的副程式與它具有相同的功能，這麼做會使您的程式較爲簡捷；其次，檢查這個組合語言副程式是否需要由BASIC程式中傳送元素；如果您必須在執行這個副程式之前先設定微處理機內暫存器的值，或是在副程式執行完畢後必須將所得到的值放入暫存器內，那麼您必須使用額外的組合語言指令與BASIC語言聯繫；雖然如此，仍然是有一些工作可以由BASIC直接來做的。

當您知道了每一個副程式的功用後，您也許要將它們的資料及功能記錄起來，使得以後使用時更加方便。例如，CALL－936會把顯示文字資料的螢幕清除乾淨，並且把游標放在螢幕的左上角。使CALL 敍述更具有說明性的一種方法就是在程式的起頭處先設定一個變數，就如下面的情形一般：

```
10 CLSCREEN=-936
```

然後在這個程式稍後的地方再使用它：

```
1510 CALL CLSCREEN
```

這麼做會使得CALL 敍述更加能夠讓人瞭解它的功能，但是却會在程式中多加一個敍述。像這種改進的方法使您的程式更容易被瞭解，而且偵錯時也方便的多。

需要避免的問題

如果您具有由編修狀態進入的組合語言程式，那麼就可以重新設定起始位置及重新組合這個程式而將它放置在適當的位置內，但是，如果您使用MINI－ASSEMBLER 設計一個機器語言的副程式，而這個副程式被設計與BASIC合併使用時，您也許會遇到一些問題使得您在遇到不同型式的APPLE Ⅱ及不同大小的記憶體位置時必須重新設計您的副程式；如果您使用的記憶體位置已經被DOS、圖形頁或由磁帶、磁碟輸入的APPLESOFT 所佔用了，那麼您可能就要被強迫重寫您的副程式了。

如果您的程式是APPLESOFT 語言所設計的，那麼當您想要與副程式之間做傳送元素的工作時，您必須使用USR函數，而不是使用CALL敍述，由9D到A3的位置內儲存著使用USR函數傳送的元素值，而且您也可以使用這個區域將元素值送回BASIC內。您必須使用10到12的位置（十六進位的ØA到ØC）放進一個JMP指令（使用POKE 敍述），因爲您若要使用USR函數使得機器語言副程式開始執行，那麼這些位置內必須放入一個JMP的指令把控制轉移到這個副程式的開始處。

使您的BASIC程式更加完善

在BASIC系統內，您可以使用LOMEM：及HIMEM 敍述將您的組合語言程式保護起來，使不致被BASIC系統所破壞；如果您希望由磁碟或磁帶輸入的APPLESOFT或DOS同時存在記憶體內，那麼在設定一個組合語言程式之前尚有許多需要注意的地方；下面就是同時將一個組合語言副程式或程式與BASIC及DOS合併使用時的一般規則：

1. 將DOS存入記憶體內。
2. 如果需要的話，則將APPLESOFT 由磁帶或磁碟輸入記憶體內。
3. 設定LOMEM：及HIMEM：值。
4. 將組合語言程式存入記憶體。
5. 將BASIC程式由磁帶或磁碟存入記憶體內（或由鍵盤輸入）。

磁碟作業系統會在您將它由磁碟存入記憶體後重新設定HIMEM：的值，而當APPLESOFT被存入時則會重新設定LOMEM：的值；因此您必須重設LOMEM：及HIMEM：的值，使得您的組合語言程式有位置可以存放，那麼隨後的BASIC程式只會對位於LOMEM：與HIMEM：間的記憶體產生影響，而不會對您的組合語言程式造成破壞。

對於尋找安全的位置供您的組合語言程式存放，您可以參考附錄G的記憶體分佈圖及第八章中對LOMEM：及HIMEM：的討論。

8

BASIC敍述與函數概要



這一章說明APPLE Ⅱ所有的BASIC敍述與函數；我們首先用字母的先後次序來解說各個敍述，接著再講解各個函數。

本章應該是整個敍述與函數的索引，而第三到第七章才是程式製作的觀念，也只有那兒才有完整的敍述與函數的例題。

直接式與間接式

大多數的敍述都能夠在直接式或間接式的方式下執行；除開了特別指明的之外，您可以假設在兩種方式之下都可以接受。當然，特異所在是會指出來的，有些敍述只能在某個方式下執行，但在另一個方式時則不能；也有些敍述能夠在兩種方式之下作業，不過在實用上來說，常常也只限於其中的一種方式而已。

這些敍述中有些是磁碟作業系統（DOS）的敍述，它們可以直接地寫成例子中的型式，在直接式之下作業；在間接式的時候，却必須要用字串的方式由PRINT敍述提出來，而第一個字元却必須是CTRL－D（ASCII碼爲4）。所以下面的兩道敍述是具有同樣的功能的：

```
]CATALOG
]PRINT CHR$(4); "CATALOG"
```

除開使用CHR$(4)之外，您也可以先打一個引號，再打一個$C_{TRL}$-D，最後是另一個引號來完成這個符號；不過在螢幕上您只能看到連續的兩個引號而已，換句話說，G_{TRL}-D這個字符是〝看不見〞的。

BASIC的版本

除非特別指明，所有的敍述與函數在整數BASIC或APPLESO-FT 中都可以使用；如果在這兩個版本中有差異的話，我們也會明白地指出來。

在後面的敍述中，我們用一個統一的格式來說明每一道敍述以及函數；首先，我們討論一些有關標點符號、大小寫字母與其它標誌的用法。

{ } 大括號表示一個可供選擇的所在；在大括號內的內容必定要選擇一個。請注意，在實際寫作敍述時，大括號不應該寫進去。

[ ] 方括號表示這是個可有可無的部份。請注意，在實際寫作敍述時，方括號不應該寫進去。

…… 連續的點表示在它前面的那一個項目可以反覆地使用；這些點也不應該寫到敍述中。

列編號 在間接式的狀況下，每一道敍述那一列的最前方必定要有個列編號，也可以說是行號。

其它標點符號　　其它的標點符號，好比說逗點、分號、引號，括號等等，必須要依照說明中的方式寫出來。

大寫字母　　大寫字母或單字必須要依照說明中的方式寫出來。

斜體英文字　　這一類是些一般性的總稱符號，您必須要依照下一段所講的意義插入適當的值或字符。

接著我們討論用在敍述與函數中的一般性總稱符號——也就是斜體英文字；如果在某個敍述中有一個斜體英文字並沒有在這兒列出來，那末這個字就對該敍述有特殊的意義，因此，您就得在那個敍述的說明中尋找它的意義了。

col　　低解析度圖形的行號；它是一個值在0與39之間的數值運算式。

colh　　高解析度圖形的行號，它是一個值在0到279之間的數值運算式。

const　　任意的數值或字串常數。

Dn　　磁碟驅動機號碼，它必須是D1或D2兩者之一。

expr　　任意的數值、關係、或邏輯（也叫做布爾——BOOLEAN——值，只能在APPLESOFT中使用）的常數、變數、或運算式；也可以是上述數種情況正確的組合式。

expr$ 任意的字串常數、變數、或運算式。

exprnm 任意的數值常數、變數、或運算式。

filename 任意的磁碟檔名稱。

line 任意 BASIC 程式中的列編號。

line$_i$ 很多 BASIC 程式中列編號的其中一個。

memadr 這是一個數值的常數、變數、或運算式，它的值被拿來當做一個記憶體位址看待。記憶體位址只能在－32767到32767之間；在 APPLESOFT 中，位址的值是從0到65535，此地－65535相當於1，－65534為2，－65533為3，……等等。

memloc 任意的以整數常數寫出來的記憶體位址，這個常數的值要在0到65535（十進位）之間；若用十六進位來寫就是$0到$FFFF之間。十六進位常數是以一個數之前是否有一個錢號（$）來識別的。

message 任意的寫在兩個引號之間的內容。

row 低解析度圖型的列號，它是1個值在0與47之間的數值運算式。

rowh　　高解析度圖形的列號；它是一個值在 0 與 191 之間的數值運算式。

Sn　　輸出、輸入接點號碼，這必須是 *S*0、*S*1、*S*2、*S*3、*S*4、*S*5　*S*6 或 *S*7 中的一個。

var　　在整數 BASIC 中，指的是任意數值或字串變數。在 APPLESOFT　，指的是任意數值、整數、或字串變數。

varnm　　任意數值變數名稱。

***var*(*sub*)**　　在整數 BASIC 時，這是有足碼的數值變數。在 APPLESOFT ，這是有足碼的整數、數值或字串變數。

Vn　　這是磁碟片識別號碼，必須是 *V*0 與 *V*255 之間的一個。

敍　述

APPEND

打開某個檔案（見 OPEN ），然後把檔案指標指到檔案的終了。

格式：

APPEND *filename* [,D*n*] [,S*n*] [,V*n*]

表 8－1 修復APPEND的機器語言程式

機器語言		6502 組合語言	
十進位	十六進位	指令	註解
169	A9	LDA $0	在$ FDED 那兒的監督副程式把A暫存器的內容（目前爲$O ）輸出到所選定的磁碟中（見附錄）
0	0		
76	4C	JMP $FDED	
237	ED		
253	FD		

這個檔案必須要是個順序檔；系統會爲它配置一個有595 個 byte 大小的緩衝區，於是 **WRITE** 指令就可以用來把資訊存到這個檔案裡頭了；儲存的所在是在第一個沒有用到的 byte 開始，除非這個檔案的中央就有沒有用到的 byte ，一般來講，這是從該檔案最後一個 byte 的下一個 byte 那兒放起的。

有時候， **APPEND** 不會從這個檔案的第一個沒有用到的 byte（通常是檔案終了所在）放起，而是從檔案的開頭存放的（可怕！）。爲了要確定這個現像不致於發生，在檔案結檔之前您就應該先寫出一個檔案終了的標誌，上面表 8－1 中列出的機器語言副程式可以爲您做這件事。首先，您可以用 **POKE** ，把這個只有五個 byte 的副程式放到記憶體中任意可以讓您使用的地方（除非您另有用途，一般來講，768 到772 這五個位址是可以用的），然後在結檔之前叫用（使用 **CALL** ）它就行了。

如果在接點 **S*n*** 的 **D*n*** 磁碟機上頭找不到這個檔案，那麼就會顯示一個 **FILE NOT FOUND**（找不到檔案）的訊息。如果在 **S*n*** 的 **D*n*** 上頭的磁碟片識別號碼不是 **V*n*** ，那麼就會得到**VOLUME MI-**

SMATCH （磁碟片不對）的錯誤。如果在 S*n* 的 D*n* 上頭的磁碟片識別號碼是V 0 ，那麼不管您寫的V*n* 中 n 是多少都不會有錯誤。如果您用的檔案已經被開啓了，那麼系統就會把它結檔（見 CLOSE ），再把它打開。

D*n* 、S*n* 與V*n* 可以用任意順序來寫。如果省略了D*n* 或 S*n* ，系統就用上一次使用磁碟機或接點的號碼代替。如果不寫V*n* ，系統就假設成V 0 。又若 n 都沒寫，系統就假定是D0 、S0 與V 0 。

APPEND是個DOS 的指令，在間接式時需要使用 PRINT 與 C TRL - D的技巧。

不應該在直接式中使用它。

AUTO

在整數 BASIC 中自動編上列編號。

格式 ：

AUTO *line* [, *increment*]

列編號從 ***line*** 這個列編號開始，每當您按下 RETURN 鍵時，列編號就自動地加上 ***increment*** 那兒的值，作爲下一列的列編號；如果不寫 ***increment***，那麼增加的值就自動地設定爲 10 。如果不再需要自動編號了，那麼可以用C TRL - X 删掉某個列編號；如果在下一列用 MAN 指令（參看 MAN的說明），則自動編號又會恢復。

這個指令只能用在直接式。

APPLESOFT 中沒有這項功能。

BLOAD

從磁碟中找出某個二進位的檔案，並且把它存放在記憶體中的某一

段位置。

格式：

BLOAD *filename* [,A*memloc*][,D*n*][,S*n*][,V*n*]

如果第二個項目（A參數）沒有寫，系統會把該程式抄到記憶體中它原先被保存起來的位置（見 BSAVE）；若有A這一項目，就把該程式抄到*memloc* 這個位置開始的區域中。

您必須要小心使用 BLOAD，因爲，當程式被抄入記憶體的同時，它所佔的區域中任何內容（比如說，資料、程式、APPLESOFT DOS 等等）都會被蓋掉。

如果這個檔案不在接點S*n*的磁碟機D*n*上頭時，就會導致FILE NOT FOUND（找不到檔案）的錯誤；又若S*n*的D*n*上頭的磁碟片識別號碼不是V*n*，那就會引起VOLUME MISMATCH（磁碟片不對）的錯誤。

D*n*、S*n*與V*n*可以用任何的次序來寫；如果D*n*或S*n*沒寫，那麼就使用上一次對磁碟機或接點的參考數值；若V*n*沒寫，就假定爲V0。

這是個DOS指令，您在間接式的時候得用 PRINT 與 CTRL－D的方式來寫。

BRUN

在磁碟上找出一個二進位的檔案（當然，這應該是個機器語言的程式），把它抄到特定的記憶體中，然後執行一道轉向（在 6502 組合語言中就是 JMP-*jump*）指令，跳到該程式的起點。

格式：

BRUN ***filename*** **[,A*memloc*][,D*n*][,S*n*][,V*n*]**

如果沒寫A那一欄，那麼這個檔案就會抄到它被保留（見 BSAVE）起來時候所在的位置；如果寫了A，就抄到*memloc*的位置。

機器語言的程式很有可能在某個特定的位置才能正常作業，因此在使用這個敍述之前，您應該仔細地檢查一下是否有一些涉及到與位址有關的指令。因爲 BRUN 會毀掉抄入記憶體後該檔案所佔的位置，所以得要小心檢查是否在這個過程中會影響到 DOS 或 APPLESOFT 。

如果在接點 S*n* 的磁碟 D*n* 上找不到這個檔案，就會引起FILE NOT FOUND（找不到檔案）的錯誤；如果在D*n* 上的磁碟片識別號碼不是V*n*，因而就會導致VOLUME MISMATCH（磁碟片不對）的錯誤。

D*n* 、S*n* 與 V*n* 可以用任何順序來寫；如果省略了 D*n* 或 S*n*，就用上一次對磁碟機或接點的參考的數字代替；若V*n* 沒寫，系統就假定爲V0 。

這一個DOS敍述，在間接式時使用需要用到 PRINT 與 CTRL-D的技巧。

BSAVE

建立一個磁碟檔，並且把APPLE Ⅱ中某一段記憶體的內容以二進位的型式保存到這個檔案中。

格式：

BSAVE ***filename*****,A*memloc*,L*length*[,D*n*][,S*n*][,V*n*]**

此地A欄的內容表示需要保存起來的那一段記憶體起點的位置：

L欄表示這一段記憶體的長度——以 byte 做單位，它的值要是 0 到 32767（十進位）之間的一個整數，當然您既可以用十進位來寫，更可以用十六進位來寫，只要數字前面加個錢號（ $ ）就行了。

如果 S*n* 上頭的 D*n* 並沒有 V*n* 這個磁碟片，就會產生VOLUME MISMATCH（磁碟片不對）的錯誤。

D*n* 、S*n* 與V*n* 可以用任何次序來寫。如果D*n* 或 S*n* 沒寫，則使用上一次對磁碟或接點參考的值；若 V*n* 沒寫，就假定爲V0 。又，*n* 也可以不寫的，這個時候就用D0 、S0 與V0 來替代。

這是道 DOS 指令，在間接式時需要用到PRINT與C_{TRL} - D的技巧。

CALL

轉向到某個位址的機器語言副程式那兒去執行。

格式：

CALL ***memadr***

CALL 敍述可以用在您自己所寫的副程式，並且更可以用來叫用很多系統中已經爲您寫好的內副程式（ *intrinsic subroutines* ），請參看附錄D中的說明。

CATALOG

把某個磁碟片上頭的檔案列出來。

格式：

CATALOG [,D*n*][,S*n*]

CATALOG 首先顯示DISK VOLUMN這個字，緊接著的是該

磁碟片的識別號碼；如果在這個敍述中寫了 V*n*，不管它的值是多少，系統都不予理會。

在磁碟片號碼下頭所列出來的就是它裡面所儲存的所有檔案的名稱；對於每一個檔案而言，CATALOG 會顯示出一個字母，表示該檔案的類型；又，這個所佔的區段（*sector*）數目，以及這個檔案的名稱。在檔案類型左邊如果有一個星號的話，就表示這個檔案是被鎖住（參看 LOCK ）了的。檔案類型的代號如下：

I：整數 BASIC 程式；
A：APPLESOFT程式；
T：文字檔案；
B：二進位（機器語言）檔案。

如果一個檔案的長度超過了255個區段（*sector*），那麼顯示出來的就是被256去除之後的餘數，也就是說，256 顯示出來的為 0，257 為 1，……等等。

D*n* 與 S*n* 的順序沒有關係，如果 D*n* 或 S*n* 沒寫，就使用上一次的值。

這是個 DOS 指令，在間接式時得要用到 PRINT 與 C_{TRL}-D 的技巧。

CHAIN

從磁碟中把某個整數 BASIC 程式抄入記憶體，並且立即執行它；但是在抄入記憶體時並不會消除任何變數或陣列的值。

格式：

CHAIN *filename*[,D*n*][,S*n*][,V*n*]

CHAIN 只能在整數 BASIC 中使用，並且也只能抄入整數 BASIC 的程式。

如果S *n* 上頭的D*n* 中沒有那個檔案，那麼就會發生FILE NOT FOUND（找不到檔案）的錯誤；若 S*n* 上的D*n* 中的磁碟片識別號碼不是V*n*，於是就會發生VOLUME MISMATCH（磁碟片不對）的錯誤。

D*n*、S*n* 與V*n* 的順序可以自由排列。如果 D*n* 或 S*n* 沒有寫，就用上一次的值替代；如果V *n* 沒寫，就定成V 0。另外，*n* 也可以省略，這個時候就假定爲D 0、S 0、V 0。

這個道 DOS 指令，在間接式中得要使用 PRINT 與 C_{TRL}-D 的技巧。

CLEAR

這是一道 APPLESOFT 的敍述，它把 0 存到所有數值變數與數值陣列的元素中，並且又把一個虛無字串（*null string*）存到所有字串變數以及字串陣列的元素中。

格式：

CLEAR

執行這道敍述的動作就相當於您把APPLE Ⅱ關機，再開機，之後把程式再抄入記憶體所得到的結果。執行過 CLEAR 之後，如果上述的動作不會影響程式的邏輯，那麼您的程式還是可以繼續執行下去的。

在整數 BASIC 時，請用 CLR。

CLR

這一道整數 BASIC 的敍述把所有數值變數以及數值陣列的元素

淸成零，並且把虛無字串（*null string*）存到字串變數中。

格式：

CLR

這一道敍述還會把所有陣列與字串的度量（*dimension*）重訂過；不過，當您執行過 CLR 之後，在把值存入任何變數之前，陣列的值還是可以印出來的，換言之，尙未消失。

CLR 只能在直接式之下使用。

在 APPLESOFT 時要用 CLEAR 。

CLOSE

系統收回該檔案的緩衝區（*buffer*），如果對該檔案的上一個動作是WRITE ，那麼就把輸出緩衝區的內容（如果有的話）存到該檔案中。

格式：

CLOSE [*filename*]

如果有一個檔案您是用 WRITE 來使用的話，於是爲了避免失掉資料，您必須要 CLOSE 它。如果您寫了 *filename*，那麼只把該檔案結檔；但若沒寫*filename* ，那麼就把所有的檔案（除了那個控制用的 EXEC 檔之外）都結檔。

在DOS 3.2.1或更早期的版本中，當某個順序檔結檔時可能會剛剛好地塡滿一個區段（*sector*）；在此情況下。下一個APPEND指令就會在這個檔案的頭頭加內容，而不是加在最後了。爲了克服這一點，在 CLOSE 敍述執行之前，您應該叫用在表 8－1 那兒的相當短的機器語言副程式；您可以先把那個副程式用 POKE 的技巧存到任何地方（只要有連續的五個 byte 就行），比如說 768 到 772 那兒一

一除非此地您另有用途。

這是個DOS指令，在間接式時需要用 PRINT 與CTRL－D的技巧。

COLOR=

在低解析度圖型作業方式時定出顯示的顏色。

格式：

COLOR = ***exprnm***

表8－2　低解析度顏色碼

編碼	顏　色	編碼	顏　色	編碼	顏　色	編碼	顏　色
0	黑	4	深　綠	8	棕	12	綠
1	紫　紅	5	灰	9	橘	13	黃
2	深　藍	6	中　藍	10	灰	14	水　色
3	紫	7	淺　藍	11	粉　紅	15	白

一直到下一道 COLOR 敍述被執行之前，所有PLOT、VLIN與 HLIN 敍述所繪出來的圖所用的顏色就由此敍述造出來。顏色的代號如表 8－2 所示；***exprnm*** 的值必須在 0 與 255之間，如果是實數值，就會先轉換成整數值，然後取該整數值被 16 除的餘數；好比說，0、16、32 的値都相當於 0。如果以前沒有定出顏色的值，就假定爲 COLOR＝0。

在高解析度圖像時， COLOR 是不起作用的。在顯示文字時，其字母在螢幕上的顏色由 PLOT 把這個字母顯示在螢幕上的位置而定；請參看 PLOT 那兒的說明。

CON

這是個整數 BASIC 的指令；當程式執行中止後，使用這道指令可以讓程式從剛才打斷的敍述的下一道敍述開始繼續執行。

格式：

　　CON

CON 是當我們使用 CTRL－C或 RESET 打斷程式執行之後，還要讓剛才被打斷程式繼續執行的時候用的。如果並沒有程式被打斷，CON 只是叫系統沒有動作而已。請注意，INPUT 敍述如果被 CTRL－C 打斷的話，這是不可以用 CON 再繼續的。

當程式打斷後，我們把程式中某列加以修改，或加上一列，或者是因爲產生了一道錯誤的訊息而停頓；有時候，CON 還是能夠繼續，不過可能會繪出一個錯誤訊息，或者是把系統鎖住。

CON 只能用在直接式。

APPLESOFT 是使用 CONT。

CONT

當程式停頓後，這一道 APPLESOFT 指令讓程式從停頓時的下一道敍述開始執行。

格式：

　　CONT

CONT 是用在 STOP 、END 或 CTRL－C 把程式執行打斷了

，但又希望繼續執行的時候用的；不過若用 C_{TRL}－C 把 INPUT 敍述打斷的話，是不能用 CONT 來繼續的。如果並沒有程式被打斷的話，或者是當程式被打斷後您修改過或加上一列，也或者是因爲產生了一個錯誤訊息，用 CONT 來繼續上述的情況就會出現 ?CAN'T CONTINUE ERROR（不能繼續）的錯誤訊息。

整數 BASIC 使用 CON。

DATA

在 APPLESOFT 中，用它來產生一串要給 READ 作爲輸入用的資料。

格式：

DATA ***const*** [,***const*** ……]

DATA 敍述可以放在程式中任何地方，而且也不一定要讓READ 敍述使用它的資料。

DATA 可以記錄數值資料也可以有字串資料。字串資料往往需要用引號括起來，除非字串中含有空格、逗號、或冒號，一般來說，是不必使用引號的；不過要注意的是，引號却不以放在字串資料中，如果有此必要，那就請用CHR$ (34) 這個函數。

DATA 後面的常數資料也不一定非要有不可，也就是說，逗號之間可以只有空格；這樣，對數值變數來說，就得到零，字串變數得到虛無字串（即" "）。

在直接式時，如果您打了這道敍述而沒有列編號，這並不算錯，也沒有錯誤訊息出現，不過 READ 敍述就無法使用這些資料了。

整數 BASIC 沒有這一項功能。

DEF FN

DEF FN 可以讓您在 **APPLESOFT** 程式中定義並且使用特殊用途的函數。

格式：

DEF FN*nvar*(*arg*) = *exprnm*

nvar 這個實數變數指明了這個函數的名稱，之後，我們就用 **FN*nvar*** 來參考這個函數。

函數是由 ***exprnm***來定義的；***arg***是個虛擬引數（***dummy variable***）的名稱，它可以（通常會）出現在 ***exprnm***這個定義式中，在 **DEF FN** 中寫這個虛擬引數與程式中任何名稱相同的變數無關，它只是作爲這個定義式的一個記號而已。

當我們使用**FN*nvar*** 這個函數時，虛擬變數 ***arg*** 的值可以使用數值的運算式、變數、或常數給出來；至於 ***exprnm*** 中如果有其它非虛擬引數的變數，那麼在使用這個函數以前，這些變數都應該先行得到一個值。請參考本章中在函數那一節對 **FN**的說明。

DEF FN 敍述必須得寫在程式中一列上頭；不過，***exprnm*** 裡頭都可以讓您使用在此函數定義式之前就已經定義出來的其它函數，所以，使用這個技巧，您就可以定出異常複雜的函數了。另外，一個函數絕對不可以直接地、或間接地使用到自己。

函數名稱 ***nvar*** 可以重複使用，這樣後面的 **DEF FN** 就重新地定義了在前面同樣名稱的函數。

整數 **BASIC** 中沒有這一項功能。

在直接式時不能使用 **DEF FN** 敍述；然而，如果**NEW**、**CLR**、或 **LOAD** 之後，您在間接式中已經執行過了 **DEF FN** 敍述，那

麼您是可以在直接式中使用它來計算的。

DEL

刪除程式中某些列。

格式：

DEL ***line*₁, *line*₂**

這個敍述把目前在記憶體中的程式列，自 ***line*₁** 那一列起到 ***line*₂** 那一列止全部刪掉。如果程式中並沒有 ***line*₁** 這一列，那麼就從比 ***line*₁** 大的那一列開始；如果程式中沒有 ***line*₂** 這一列，則就刪到比 ***line*₂** 小的那一列爲止。

DEL 後面必須要有兩個列編號，用逗號分開來，兩個列編號都不可以小於零，第二個列編號必須要等於或大於第一個列編號。如果兩個列編號相等，那麼（最多）就刪掉一列。

在整數 **BASIC** 中 **DEL** 只能用在直接式。

如果在間接式中用到了 **DEL**（只能在 **APPLESOFT** 中使用），那麼把該刪的刪掉之後，程式就會停下來，但是 **CONT** 却無法讓程式繼續。

DELETE

刪掉磁碟中的某個檔案。

格式：

DELETE ***filename*[,D*n*][,S*n*][,V*n*]**

這道敍述把您指明的那一個檔案從磁碟中刪掉。

如果該檔案不在 **S*n*** 的**D*n*** 中，就會發生**FILE NOT FOUND**（找不到檔案）的錯誤；如果**S*n*** 的**D*n*** 中磁碟並非**V*n***，那麼就會導到**VOLUME MISMATCH**（磁碟片不對）的錯誤。

D*n* 、**S*n*** 與**V*n*** 可以用任意次序來寫。如果**D*n*** 或**S*n*** 沒寫，就用上次對磁碟或接點的值替代；如果沒寫 **V*n***，就假定成V0；另外，***n***是可以省略的，這個時候就假定爲D0、S0與V0。

這是道**DOS** 指令，在間接式中需要用 **PRINT** 與 **C**TRL **-D**的技巧。

DIM

在記憶體中爲陣列或字串保留出位置。

因爲整數 **BASIC** 與**APPLESOFT** 中有很大差異，所以下面把它們分開來說明。

整數BASIC格式：

DIM ***var*(*sub*)[,*var*(*sub*)……]**

整數 **BASIC** 中，只有一度空間的數值變數與簡單的字串變數才可以用這道敍述留位置。

當用這道敍述來留位置時，系統會依 ***sub*** 的值加上一之後的量在記憶體中劃出一塊位置；這個位置的編號自0起一直編到 ***sub*** ；第0個元素就相當於具有同樣名稱的簡單變數，也就是說，A(0)就是A。

用在字串變數時，**DIM** 敍述指出該字串的最大長度，此地，***sub*** 就代表該字串的長度。

DIM 敍述中每一個足碼 ***sub*** 的值一定要在1與255之間，除了這點之外，一個陣列的容量只會受到您記憶體容量大小的限制。

如果您用足碼參考陣列中某個元素，但足碼值却不在DIM敍述所定出來的範圍中，那麼就會出現 ***RANGE ERR 的訊息；在一個字串變數中，如果您用到比定出來長度還大的字元，於是就會產生 ***STRING ERR 的訊息。

當程式執行時，在整數 BASIC 中的DIM敍述不會把某個特定的值存到每一個陣列中，所以當造出了陣列的大小之後，把該陣列清理一番（比如說，每個元素都存入0）的工作是要您自已來完成的。另一方面，就字串變數而言，當您定好長度之後，它一定會得到一個虛無字串做開始。

Applesoft格式：

DIM ***var*** (***sub*** [, ***sub*** ……]) [, ***var*** (***sub*** [, ***sub*** ……])……]

DIM 敍述指出陣列的各個維數如下：

var (sub_i)	一度空間（或一維）陣列；
var (sub_i, sub_j)	二度空間（或二維）陣列；
var (sub_i, sub_j, sub_k ……)	多度空間（或多維）陣列。

APPLESOFT 提供了三種類型的陣列，包括整數、實數與字串等；陣列中每一個元素的類型就由該陣列的名稱來決定，至於陣列的維數就由 DIM 敍述中足碼的個數決定，當參考到一個陣列的元素時，每一個足碼的值都必須要在0到 ***sub*** 之間，此地的 ***sub*** 是 DIM 敍述中對應維數的足碼值。

至於陣列中維數的數目是受到了您究竟有多少記憶體限制的；一個陣列最多只能有 88 度空間（維數），而這只有當大多數的足碼都是0

的時候才有可能；如果 DIM 敍述中的維數等於 89 或更多，或者是大到超過了既有的記憶體容量，那麼就會導致 ?OUT OF MEMORY ERROR （記憶體不夠用）的錯誤訊息出現了。

如果您嘗試使用某個陣列的元素，但是其中某個足碼的值超過了既定的大小，或者是足碼數目與既定的數目不符，那麼就會產生 ?BAD SUBSCRIPT ERROR（不正確足碼）。

如果在 DIM 的定義之前就使用到了某個陣列，APPLESOFT 就假定這個陣列的每個足碼為 10 （即 0 到 10 ），於是就相當於在 DIM 敍述中寫上這個陣列的名稱，然後每個足碼都是 10 。

陣列的維數絕對不可以定兩次或兩次以上，縱使是其中的一次是系統代您定出來的也是一樣（見上一節），如果您嘗試重新定義一個陣列，您就會得到 ?REDIM'D ARRAY ERROR （重新定維數）的錯誤訊息。

DRAW

這是一道 APPLESOFT 的敍述，它在高解析度圖像的方式下在螢幕上繪出一個造型。

格式：

DRAW *exprnm*[AT *colh*, *rowh*]

需要繪出來的造型依 *exprnm* 的整數值來決定，然後使用上一次執行 HCOLOR 所定出來的顏色在螢幕上畫出來；這個造型的比例尺（*scale*）以及旋轉的角度可以在 DRAW 指令執行之前，用SCALE 與 ROT 指令先行定出。

圖型的造型是從 *colh* 與 *rowh* 整數值座標那兒開始畫起，但若您沒有寫出這些值，那麼就從上一次執行DRAW、XDRAW或HPLOT

指令時的最後一個點那兒開始。

造型的號碼（由 ***exprnm*** 決定）必須要在 0 與造型表（ ***shape table*** ）的大小（不可以大過255 ）之間。

如果記憶體中根本沒有造型表，那麼請避免使用 **DRAW** ，因爲系統可能會鎖住而沒有任何動作，或者是螢幕上出現一些亂七八糟的圖型。另外，如果您的程式佔用了記憶體中留給高解析度圖像區域的話，那麼您很可能毀了這個區域的一部份，甚至於全部。

不能在整數 **BASIC** 中使用。

DSP

當整數 **BASIC** 在執行時，顯示某個特定變數所改變的值。

格式：

DSP ***var***

當指出的某數的值改變時，就會顯示出該變數新近得到的值，並且把對應的列編號也一併顯示出來。請小心，這個指令所顯示的資料會與您程式的輸出混雜在一塊，以致於很難閱讀。 **RUN** 指令會消除 **DSP** 指令，所以當您在直接式中用 **DSP** 來除錯時，請使用 **CON**或**GOTO** 來繼續您的程式。

如果您不想要 **DSP** 時，就打入 **NO DSP** 。

APPLESOFT 中沒有這道指令。

END

造成程式執行停頓。

格式：

END

停頓的時候不會顯示任何訊息。在整數 BASIC 中，END必須要是最後一個執行的敍述，要不然就會發生 ***NO END ERR（沒有 END ）的錯誤；不過，在 APPLESOFT 中，有沒有END 都沒關係。

整數 BASIC 在直接式時不能用它。

EXEC

如果磁碟中的某個檔中文字全都是由鍵盤輸入的話，那麼就可以用此指令"執行"這一個檔案。

格式：

EXEC *filename* [,R*n*][,D*n*][,S*n*][,V*n*]

由 EXEC 所使用的文字檔可以包含有 BASIC 指令，DOS 指令，以及程式列。當執行這個檔案時，首先從磁碟中讀入這個檔案的第一列，如果它是道指令，就馬上執行它，如果是道程式列，就加到記憶體的程式中，這與您從鍵盤上鍵入這一列的效果是完全一樣的。

EXEC 檔可以讓我們寫進去一個完整的程式，把它列出來，加以執行，再保留到磁碟中，修改程式，以至於任何鍵盤上可以做的動作都能夠完成。您更可以用一個 EXEC 檔來產生，並且執行另一個EXEC檔。

如果有寫R這一欄，它指出第一個被執行檔中開始執行的欄位；EXEC 檔的欄位，就R來說，永遠是從0開始算的，因此第一欄是0，第二欄是1，第三欄是2，……等等。R後面的數必須是0與32767

之間的整數，如果R*n*的值恰好是檔案終了的下一欄，那麼什麼事都不會發生；如果它指出的位置在檔案終了的後兩欄，或者是更後面的欄位，就會發生END OF DATA（資料終了）的錯誤。

當某個 EXEC 檔正在執行，但是一個 INPUT 敍述却要讀入資料時，那麼資料將會從 EXEC 檔中取得。

EXEC 檔中最後一列被執行完了之後，它會把自己結檔（見 CLOSE ）；如果在 EXEC 檔執行的過程中，又碰到另一 EXEC 指令，那麼系統會先把原來的先結檔，於是在原來檔中任何剩餘的指令就都不予理論，接著再把新的 EXEC 檔開檔，並且執行這個新檔中的指令。

如果S*n*的D*n*中沒有這個檔，就會發生FILE NOT FOUND（找不到檔案）的錯誤；如果S*n*的D*n*中不是V*n*，就會引起VOLUME MISMATCH（磁碟片不對）的錯誤。

D*n*、S*n*與V*n*可以用任何次序來寫；如果省略了D*n*或S*n*，就用上一次的值替代；如果V*n*沒寫，就假定爲V0；另外，*n*是可以不寫的，這個時候就假設爲D0、S0或V0了。

這是一道DOS指令，在間接式中要用 PRINT 與 C_{TRL}-D的技巧來寫。

FLASH

這是一道 APPLESOFT 敍述，它把螢幕上的字弄得一黑一白地閃爍。

格式：

```
FLASH
```

這道敍述執行後，會造成任何以後再執行的 PRINT 敍述顯示出

來的結果，以黑底白字、白底黑字地輪番出現；縱使是錯誤訊息也是一樣地受到影響；不過，因爲 INPUT 敘述而要您鍵入的資料在螢幕上却不受影響，另外一個不受影響的就是在此敘述執行之前就已經顯示出來了的結果。

FLASH 的結果是經由ASCII碼稍加修飾之後而達成的，因此在閃爍的狀況下任何保留到磁碟上的字元事實上都不是正確的字碼；當您把它再讀回來時，就會得到錯誤的字母了。

整數 BASIC 中沒有此一功能。

FN

在本章有關函數的討論中您會看到詳細的說明；您也可以參看 DEF FN。

FOR

這個敘述開始執行一組需要重複執行的敘述（就是迴圈，*loop*），一直到某個變數的值增加到某一特定量爲止。

格式：

FOR *varnm* = $exprnm_1$ TO $exprnm_2$ [STEP $exprnm_3$]

一執行到 FOR 的時候，*varnm* 就得到 $exprnm_1$ 的值，接著就執行 FOR 以後的敘述，一直到遇見 NEXT 敘述爲止，於是 *varnm* 就加上 $exprnm_3$ 的值（如果不寫 STEP，那麼它的值就假定爲 1）。於是新的 *varnm* 值就拿來同 $exprnm_2$ 的值相比較，比較的方式由 $exprnm_3$ 值的符號來決定；如果它的符號爲正，就看：*varnm* 是否

小於或等於 ***exprnm***$_2$，若然就回到 **FOR** 敍述的下一條敍述執行；若 ***exprnm***$_3$爲負值，就比較看 ***varnm*** 是否大於或等於 ***exprnm***$_2$，如果是，也跳回到 **FOR** 的下一條敍述去執行。如果比較不成立，即 ***varnm*** 大於 ***exprnm***$_2$（若 ***exprnm***$_3$大於零）或 ***varnm*** 小於 ***ex-prnm***$_2$（若 ***exprnm***$_3$小於零），則執行 **NEXT** 的下一道敍述。因爲這個比較是在把 ***varnm*** 的值更動了之後才做的，所以在 **FOR** 與 **NEXT** 之間的敍述必定會至少執行一次。

整數 **BASIC** 中 ***varnm*** 必須要是整數變數，而 **APPLESO-FT** 中 ***varnm*** 必須是實數變數；它絕不以是字串變數。

由 ***exprnm***$_1$、***exprnm***$_2$、***exprnm***$_3$所決定出來的開始值、終了值與增加值只有在第一次執行到 **FOR** 時算一次，因此在廻圈中若您改變了構成這三個式子中的某個變數，就不會對廻圈的執行有任何影響；但是，您可以改變 ***varnm*** 的值，那麼您就可以改變了原來應該循環的次數了，比如說，若您把 ***exprnm***$_2$存入 ***varnm***，因此下一次循環就不會繼續了。請注意，不要在副程式之外開始一個廻圈，但却在副程式之內來結束它。

FOR-NEXT 廻圈也可以互相套疊在一塊兒，這樣套在一起的廻圈的 ***varnm*** 都必須得相異，而且每一個廻圈都得完全地包含在包圍它的廻圈之內；不過，廻圈的終結却可以是同一所在。在整數 **BASIC** 中，套起來的廻圈可以有十六層，但 **APPLESOFT** 中却只能有十層。

只有在 **APPLESOFT** 時才可以在直接式之下使用 **FOR** 敍述，不過整個廻圈却必須要寫在同一列上頭；如果沒有寫 **NEXT**，那麼廻圈只會執行一次。

FP

讓您的 **APPLE Ⅱ** 進入 **APPLESOFT**。

格式：

FP [,D*n*][,S*n*][,V*n*]

APPLESOFT 的來源取決於您擁有的APPLE Ⅱ以及加裝了些什麼物事：

1. 如果您有的是APPLE Ⅱ PLUS，語言是在ROM中，因此就不管您還有什麼設備了。
2. 如果您有一片APPLESOFT卡，FP就從那兒取得該語言，而不理會卡上的開關是如何設定的。
3. 如果用的是*Apple Language System*，那麼FP就從那兒取得APPLESOFT。
4. 對於其它APPLE Ⅱ來說，FP就從您指明的磁碟上去找APPLESOFT，如果在那兒找不到，就會出現LANGUAGE NOT AVAILABLE（沒有這套語言）的訊息。

FP會把記憶體中的任何程式洗掉。

如果S*n*上的D*n*中沒有這個檔案，就會產生FILE NOT FOUND（檔案找不到）的錯誤；如果S*n*上D*n*中並不是V*n*，於是VOLUME MISMATCH（磁碟片不對）就會出現。

D*n*、S*n*與V*n*可以用任意次序來寫；如果省略了D*n*或S*n*，就用上一次的值替代；如果沒寫V*n*，就假定爲V0。另外，*n*也可以不寫，於是就使用D0、S0或V0。

只能在直接式使用。

GET

這是一道APPLESOFT的敍述，它從鍵盤上接受一個符號，但

是却不會在螢幕上顯現出來。

格式：

GET ***var***

執行到這個敍述之後，機器就會暫停，一直到某個鍵被按下去爲止。若 ***var*** 是個字串變數，則按下去那個鍵的符號就存入 ***var*** ，如果鍵入的是 C_{TRL} - @，那麼存入該變數的就是虛字串。

GET 通常是不會與數值變數 ***var*** 連用的；如果它果然是數値變數，則若鍵入的是 0 到 9 十個鍵其中的一個，就會把對應的數值存入該變數。如果鍵入的是加號、減號、逗號、冒號（ : ）、C_{TRL} - @ 、E、或者是句點，那會就把 0 存入該變數；除此之外，任何鍵都會得到 ?**SYNTAX ERROR**（語法錯誤）的訊息，從而程式就停頓了。

GET 不能用於直接式。

整數 **BASIC** 中無此功能。

GOSUB

這道敍述使得程式的執行轉向到所指定的那一列，當執行到 **RET-URN** 敍述時，就回轉到 **GOSUB** 的下一道敍述。

一般格式：

GOSUB ***line***

GOSUB 敍述叫用某個副程式，這個副程式的入口就是 ***line*** 所指明的列編號。邏輯地說，副程式的入口正是副程式的起點，也就是說，正是包含了第一個被執行到敍述所在的那一列；不過，入口那一列的列編號却不一定是這個副程式中列編號最少的那一列。

當執行完這個副程式之後，執行就會轉回到 GOSUB 敍述的下一列，每一個副程式都可以使用 RETURN 敍述來完成把執行轉回去的要求。

GOSUB 敍述可以寫在程式中任何所在，於是，這意味著程式在任何所在都可以叫用一個副程式。

副程式也可以套疊起來；這是說，在副程式中也可以叫用副程式；在 APPLESOFT 中容許有二十五層，換言之，在任何 RETURN 敍述被執行之前，可以有二十四道 GOSUB 敍述被執行；整數 BASIC 中却只有 16 層 GOSUB 。

一般而言，您必須用 RETURN 來離開副程式，而不是使用GOTO 敍述；不過，如果您先執行一道 POP 敍述，您是可以用 GOTO 跳出副程式的。

在整數 BASIC 時不能在直接式之下使用。

整數BASIC中的另一格式：

GOSUB *exprnm*

整數 BASIC 中容許您把 *line* 用一個運算式來替代；如果*exprnm*的值不與任何列編號相等，就會顯示 ***BAD BRANCH ERR （轉向錯誤）的訊息。這道敍述使您在整數 BASIC 中模擬 ON-GOSUB 的動作。

GOTO

沒有條件地使程式的執行轉向到某一指定的列。

一般格式：

GOTO *line*

程式的執行立即進入 ***line*** 那一列的第一道敍述去執行，如果程式中並沒有這一列，**APPLESOFT** 就會顯示出 **?UNDEF'D STATEMENT ERROR**（沒有定義的敍述）的錯誤訊息；在整數BASIC 中却是出現 *****BAD BRANCH ERR**（轉向錯誤）。

整數BASIC中的另一格式：

GOTO *exprnm*

在整數 **BASIC** 中是可以用一道運算式來取代列編號的，如果程式中並沒有 ***exprnm*** 這個値的列編號，就會顯示出錯誤訊息 ***** BAD BRANCH ERR**（轉向錯誤）。這個型式的 **GOTO** 可以讓我們在整數**BASIC** 中模擬 **ON-GOTO** 的動作。

GR

把螢幕轉換成低解析度的圖像（40 × 40），並且把最下面的四列留作文字的輸出。

格式：

GR

螢幕上圖形區域被清成黑色，而游標移到文字區域（又叫做文字幕——***text window***），另外**COLOR**也定成 0（黑色）。

如果執行這道敍述時，**HGR** 正在作用，**GR** 的動作如上；不過，若 **HGR2** 正在作用，就會讓您看到低解析對圖形與文字的第二頁，這會叫您有點困擾，因爲螢幕上會顯現一些亂七八糟的內容，而您鍵入的資料却看不到。爲了要回到文字的方式，您可以鍵入 **TEXT**；所以，從 **HGR2** 轉到**GR** 之前，您的程式應該先要執行 **TEXT**。

當GR執行完之後，您可以用POKE - 16302,0 轉變成整個螢幕（40 × 48）的低解析度圖形去；此後，您在直接式之下鍵入的任何內容，都會在最下面的四列以顏色點的方式顯現出來，不過所有的執行也自然是正確的。再用POKE - 16302, 0就會還原成40 × 40。

HCOLOR=

這是道APPLESOFT的敘述，它在高解析度圖像方式中定出繪圖所用的顏色。

格式

HCOLOR= ***exprnm***

一直到下一道 HCOLOR 敘述之前，所有 HPLOT 與 DRAW 敘述都用所指定的顏色來繪圖；表 8 － 3 列出各種顏色的代號，請注意到 ***exprnm*** 的值必須要在 0 到 7 之間，如果超出了這個範圍，就會導致 ?ILLEGAL QUANTITY ERROR（不正確數值）的訊息出現；另外，在第一道 HCOLOR 敘述執行之前，在高解析度圖像控制之下繪出來的圖，所用的的顏色是無法預先知道的。

表 8 － 3 高解析度顏色碼

編碼	顏色	編碼	顏色
0	黑	4	黑
1	綠	5	橘 *
2	紫 *	6	藍 *
3	白	7	白

* 依TV的控制鈕而定

HCOLOR 對低解析度圖像不會構成任何影響，而且當APPLE Ⅱ並非在高解析度圖像控制之下去執行 **HCOLOR** 敍述也不會影響到下一個高解析度圖像繪圖時所用到的顏色。

整數 BASIC 中無此功能。

HGR

這一道 APPLESOFT 敍述把螢幕轉換成高解析度繪圖（ 280 × 160 ）的方式，在最下方並且留出四列作爲文字顯示之用。

格式：

HGR

這會顯示高解析度螢幕記憶區域的第一頁，不過不會對低解析度（文字）的螢幕區域有所影響，但是却只能讓您看到最下方的四列。游標是不會移到這四列的區域中的，而且當執行過 HGR 之後，直到您鍵入若干列之前，很可能您都看不到它。螢幕會被清成黑色，另外，這道敍述並不會改變 HCOLOR 的結果。

您可以在執行過 HGR 以後，使用POKE -16302，0 把螢幕轉換成 280 × 192 的高解析度顯示方式；於是，任何您鍵入的直接式指令都看不到了，不過却還是能夠正確無誤地執行的。另外，執行POKE -16301，0 就能夠叫回這四列文字列。

如果您的APPLE Ⅱ 的記憶體少過 32 K ，您就不能同時使用 DOS 與 HGR 了，因爲它們二者會使用到相同區域的記憶體。

進一步來說，因爲不論是從磁碟或卡帶抄入的 APPLESOFT 都會佔用記憶體中第一頁裡頭高解析度圖像區域的一部份，所以在這種情況下您也就不能使用 HGR 了。

• 縱使是您有了 APPLESOFT 卡，但是如果您的程式太長了，您

的程式也有可能延伸到高解析度圖像區域的第一頁的；當然這並非不可補救，您可以用 HIMEM: 8192，這一道敍述來保護著第一頁。

整數 BASIC 中無此功能。

HGR 2

把螢幕轉換成 280 × 192的高解析度圖像方式，把高解析度螢幕記憶區域的第二頁顯示出來。

格式:

HGR 2

這道敍述對低解析度（文字）螢幕記憶區域不會有任何影響。雖然您看不到鍵入的內容，不過鍵入的指令還是能夠正常地執行的。螢幕會清成黑色，另外 HCOLOR 的結果不受此敍述影響。

若您的記憶體小於 24 K，那麼螢幕記憶區域的第二頁就不能用了；在這樣的系統中，請在執行 HGR 2 之前用HIMEM: 16384保護著這個區域，以及您的程式、變數等等，以免混淆使用同一區域。

除非您至少有 36K 記憶體，您是不能同時使用 HGR 2 與 DOS 的，換言之，磁碟 APPLESOFT 需要至少有 36K 才能用HGR 2。

請不要用POKE -16301，0來留出四道文字列，因爲這會顯示低解析度圖像的第二頁，而您的直接式指令都是送到第一頁去的，於是您就見不到鍵入的指令了（雖然能夠正確地執行）。

整數 BASIC 中無此功能。

HIMEM:

設定您的 BASIC 程式，包含變數所能使用記憶體的上限。

格式：

HIMEM: ***exprnm***

HIMEM: 建立您的 **BASIC** 程式在**RAM**所能使用的上限，而 **DOS** 是永遠被擺在 **HIMEM:** 之上的——如果有的話。使用**HIMEM:** 這道敍述，您可以爲機器語言的副程式、高解析度圖像的造型表（***shape table***）留出額外的空間，您還可以保護住**RAM** 中的高解析度圖像的螢幕記憶區域。

首先，**HIMEM:** 是造成您的**APPLE Ⅱ**中最大的位址的（比如說，48K 的**APPLE Ⅱ**就是 49151）；因爲 **DOS** 在記憶體的最高位置，因此當您把它抄入記憶體時，就會修改 **HIMEM:** 的值，把它向下推大約 10,800 個 byte；您用**MAXFILES**爲檔案留緩衝區時，每一個額外的緩衝區會把 **HIMEM:** 向下移 595 個 byte 。如果您的 **APPLESOFT** 程式使用了字串，那麼就從 **HIMEM:** 這個位置向下擺起，請參看附錄**G**的說明。

exprnm 的值必須要在－65535 與 65535（整數 **BASIC** 中是－32767 到 32767）之間，要不然就會出現錯誤訊息。

您不應該把 **HIMEM:** 的值定得比您所擁有的記憶體還大；如果這樣做了，某些變數的記憶位置可能存在不存在的地方了。

您可以用下面的技巧來看看目前 **HIMEM:** 的值：

PRINT PEEK(116)*256 + **PEEK**(115)

APPLESOFT 用

PRINT PEEK(77)*256 + **PEEK**(76)

整數**BASIC** 用

NEW、**RUN**與**CLEAR**均不會改變**HIMEM:** 的值。

如果您把 **HIMEM:** 定得比 **LOMEM:** 還小，或者是造成不夠記憶體來執行您的程式時，就會引出錯誤訊息。

整數 **BASIC** 時只能用在直接式。

HLIN

在低解析度圖像方式之下，在螢幕上描出一條水平直綫。

格式：

HLIN col_1, col_2 **AT** *row*

這條直綫是描在指定的那一列上頭，從 col_1 描到 col_2 ，所用的顏色由上一次 **COLOR** 敍述所決定；如果螢幕是顯示文字，或者是有文字幕，而且 *row* 的值大於 39 ，於是 **HLIM** 就會在文字幕那兒，描出一條由文字組成的直綫，所用的文字由上一次 **COLOR** 敍述所決定；請參看表 8 － 5（大約在本章討論 **PLOT** 敍述附近）。

整數 **BASIC** 中，col_1 必須要小於 col_2 ，要不然 *****RANGE ERR**（範圍錯誤）的訊息就會顯示出來了。

HOME

這一道 **APPLESOFT** 敍述把螢幕清掉，再把游標移到左上角（第一列第一行）。

格式：

HOME

在整數 **BASIC** 中，請用 **CALL** －936 。

HPLOT

這道 **APPLESOFT** 敍述在高解析度圖像的螢幕上頭點出一個有顏色的點或一條有顏色的直綫。

格式：

HPLOT ***colh, rowh***
HPLOT TO ***colh, rowh***
HPLOT $colh_1$, $rowh_1$ **TO** $colh_2$, $rowh_2$ [**TO** $colh_3$, $rowh_3$ ……]

上面第一種型式的指令在螢幕上指定的位置點出一個有顏色的點，這個點的顏色由上一次執行的 **HCOLOR** 決定。

第二種型式從上次停下來的一點起到 ***colh*** 與 ***rowh*** 這一個點之間描出一條有顏色的綫段，如果從 **HGR** 或 **HGR2** 兩個指令的執行到目前還沒有描述任何的點，那麼就不會畫出任何綫段或點。這個綫段的顏色由上一次執行的 **HCOLOR** 來決定。

第三種型式也是畫出一條有顏色的直綫；如果您有 **APPLESOFT** 卡，這條綫可以有好幾個綫段，首先是從 $colh_1$ 、 $rowh_1$ 畫到 $colh_2$ 、 $rowh_2$ ，綫段的顏色由上一次執行的**HCOLOR**所決定。

在磁帶與磁碟的 **APPLESOFT** 中只能有一條綫段，方括號中的內容是不能寫的；在 **APPLESOFT** 卡的時候，就會從 $colh_2$ 與 $rowh_2$ 到 $colh_3$ 與 $rowh_3$ 再描一個綫段，……等等，這些座標點可以有好多組，只要能夠寫在一列上就行了。

在文字幕內的點或綫段的一部份是看不見的，不過如果您用**POKE -16302，0** 變成整個螢幕的圖像方式時，您就可以見到它們了。

在 **HPLOT** 之前您必須先執行 **HGR** 或 **HGR2** ，要不然您可能

會摧毀了您的程式或變數。

整數 BASIC 中無此功能。

HTAB

這一道 APPLESOFT 敍述把游標移到目前所在這一列的某特定位置。

格式：

HTAB ***col***

把游標向左或向右移到 *col* 這一行，但是不會洗掉所經過的任何已經顯示出來的文字；行的編號是從左而右，自 1 編到 40 。

在整數 BASIC 中請用 TAB 。

IF-THEN

有條件地造成程式的執行轉向到某一特令的指令或某一群指令；下面有關的說明是分開成兩部份（整數 BASIC 與 APPLESOFT ）來講的。

整數BASIC格式：

IF ***expr*** **THEN** ***statement***

IF ***expr*** **THEN[GOTO]** ***line***

在 IF-THEN 第一種型式中，運算式的部份構成一個條件，如果為眞，就執行 THEN 後面的敍述；若條件爲假，就執行 IF-THEN 的下一條敍述，這個時候 THEN 後面的敍述就不會執行到了。

第二種 **IF-THEN** 型式（又叫做條件轉向）是這樣的，當條件爲眞時，程式的執行就跳到所指明的列編號那兒。

這道敍述中的運算式最常見的就是個比較式；在整數 **BASIC** 中，字串只能拿來比較相等或不相等。

expr 也可以是個邏輯値的數値的運算式，此時若 ***expr*** 的値不是零，就當成"眞"看，而零就視爲"假"。

不過 **IF-THEN** 中的運算式却絕對不可以是字串運算式（也就是說，任何會算出個字串結果的式子）。

Applesoft格式：

IF ***expr*** **THEN** ***statement***[: ***statement*** ……]

$$\textbf{IF}\ \textit{expr}\ \begin{Bmatrix} \textbf{THEN} \\ \textbf{GOTO} \\ \textbf{THEN GOTO} \end{Bmatrix}\ \textit{line}$$

第一種 **IF-THEN** 中，運算式就構成一個條件，如果它是"眞"，就執行同一列上頭 **THEN** 後面的每一道 ***statement*** ；如果這個條件有假，就跳到這道敍述的下一道敍述去執行，於是 **THEN** 之後的每一道 ***statement*** 就都被跳過去了。

第二種型式（又叫做條件轉向）是這樣的，當條件爲眞時，就跳到 ***line*** 那兒的那一列執行，要不然就繼續執行 **IF-THEN** 之後的第一道敍述。

如果 **THEN** 之後的敍述中有無條件轉向的敍述，那麼它必須是一列上的最後一道敍述，而且還得要是 **GOTO** ***line*** 的型式。如果這道無條件的轉向並不是該列上的最後一道敍述，那麼同一列上，這一道敍述之後的敍述就永遠不可能被執行到了。

IF-THEN 中式子最常見就是比較式；如果拿字串來比較，那麼就使用這兩個字串中各個文字的 **ASCII** 碼（見附錄 **I** ）來決定相對應

的大小順序。手續是這樣的，我們自左而右地一個接一個文字比下去，一直比到某個不相等的文字爲止，於是包含了ASCII碼較大的文字的字串就算是大的一個。如果沒有不相等的文字，長度較長的就算是大的一個。

這個運算式也可以是道單純的運算式，這個時候若它的值不是零，條件就算是"眞"，若值爲零，就視爲"假"，其餘的動作完全相同。

在APPLESOFT中，*expr* 還可以是一道字串運算式，不過程式中如果執行了這樣的敍述兩次以上的話，是會產生？FORMULA TOO COMPLEX ERROR這樣的錯誤訊息的。

請特別注意一點，在APPLESOFT中，若THEN字的左邊一個字母（不管中間有否空格）是A的話，就會把這個A與下一個字母合併成AT，這恰好是個保留字，因此您也許就會有麻煩了；避免之道並不難，您用個括號把式子括起來（包括了A）就行了。

如果THEN之後有一個FOR-NEXT廻圈，那麼整個廻圈都得寫在IF-THEN的同一列上頭才行。另外，THEN之後也可以有其它的IF-THEN敍述，也是只要全部都寫在原來IF-THEN所在的那列上頭就可以了。不過，使用邏輯式子卻會讓人感到更清晰，比如說，下面的兩道敍述是完全相同的，但是第二道却顯然地容易看得懂。

```
10 IF A$="X"THEN IF B=2 THEN IF C
   > D THEN 50
10 IF A$="X"AND B=2 AND C > D THEN
   50
```

IN#

選擇一個周边裝置的接點，以便以後的輸入都從該接點取得。

格式：

IN# ***slot***

以後的 INPUT 敍述都從所指明接點上的週邊裝置讀取。 ***slot*** 的值必須是 0 到 7 之間的一個整數，請注意，第 0 號接點並不是週邊裝置，因此 IN# 0 說的是使用鍵盤作爲輸入用。如果在指定的接點上頭却沒有週邊機具，那麼整個系統就會鎖住不會有任何動作，一直到您按下RESET 爲止。

如果DOS 在APPLE Ⅱ的記憶體裡頭，那麼 IN# 就看成 DOS 指令，所以在間接式中用此指令時就需要使用 PRINT 與 CTRL -D的技巧了。

INIT

啓用一個磁碟片。

格式：

INIT ***filename*** **[,Dn][,Sn][,Vn]**

這道敍述會把目前在記憶體內的程式保存到指明的磁碟片內，於是此程式就變成日後的歡迎程式（ ***greeting program*** ），一旦開機，這個程式就會自動地被抄入記憶體，並且執行。這個被啓用的磁碟片會得所指明的磁碟片編號；如果沒寫，那麼就用 254 替代。

D*n* 、**S*n*** 與**V*n*** 的順序可以隨意寫。如果沒寫**D*n*** 或**S*n*** ，就用上一次的值代替；另外，***n***是可以不必寫的，此時就假定爲D0 或S0。

INIT只能用在直接式。

INPUT

從鍵盤或其它的輸入機具接受一些字母，經過轉換之後，存入指明的一個或數個變數中。

整數BASIC中的另一格式：

INPUT [“*prompt*”,] *var* [,*var*……]

在整數 BASIC 中， INPUT 敍述要求您提供一個或多個值（可以是數值，或是字串，或是兩者的混合）；如果第一個需要資料的變數是個整數，那麼在目前游標所在的位置就會出現一個問號，作爲提示之用；不過，若第一個變數是字串變數，那麼這個問號就不會顯示出來。

prompt 是可有可無的，它是一個字串常數。如果您寫了它，系統就會把它在輸入第一個變數之前就顯示出來，不過不會在輸入每個變數之前都重覆出現一次。如果第一個變數是整數，那麼 ***prompt*** 之後就會出現個問號；若爲字串變數，那就只會顯示 ***prompt*** 而已。請注意， ***prompt*** 之後跟著的是一個逗號，並且 ***prompt*** 本身不可以是字串變數或字串算式。

如果一道 INPUT 敍述要好些個整數資料，您可以一列一個地鍵入，每個值之後按下RETURN鍵；整數 BASIC 會在每一列之後顯示一個問號作爲提示，當然，您也可以一列上鍵入好幾個數，只要兩兩之間用逗號分開就行了。

數值的輸入必須要用數字符號構成，這包含了0到9這十個符號，還有空格、正號與負號等；當需要一個數值變數時，您只按下RETURN的話，您就會收到一道錯誤訊息。

就字串變數而言，您必須每一個字串變數一列地輸入；所有（除了 CTRL－C 、 CTRL－M 、 CTRL－H 、 CTRL－U 以及 CTRL－X 之

外）的符號都可以作輸入用，從第一個符號起到按下RETURN之前的字串就會存入對應的字串變數中。當一個字串變數需要資料時，您馬上按下RETURN鍵，則此字串變數就會得到個虛無字串（即“ ”）。如果您鍵入了不合用的符號（比如說，在數值變數時鍵入文字符號），您就會得到一道警告 ***SYNTAX ERR （語法錯誤），接著再顯示RE-TYPE LINE （再來一次），您只需要把所有的資料重新鍵入就行了。

在直接式時不可以使用。

APPLESOFT格式：

INPUT [“*prompt*”;] *var* [,*var*……]

INPUT 可以要求任何的數值與字串變數的值作輸入之用；一般來說，在目前游標所在的位置都會出現一個問號，作提示用，不過若您寫了 ***prompt*** ，那麼APPLESOFT 就會省略這個問號了。

Prompt 可有可無，不過一定得是個字串常數；如果您寫了，就會在要求第一個變數值之前就會顯示出來，不過以後的變數就不會重覆出現了。請注意，有了 ***prompt*** 之後就不會顯示問號了，還有IN-PUT中***prompt*** 之後跟著的是個分號。

一般而言，如果一道 **INPUT** 敍述要求多於一個輸入資料時，您是可以一列一個地鍵入的，只要每個資料之後接著按RETURN就行了，而 **APPLESOFT** 會在每要一個值時顯示兩個問號（ ？？ ）。當然，如果需要，您也可以一列上頭打好幾個數，兩兩之間用逗號分開就行了。

如果您鍵入了不能被接受的資料（比如說，數值的資入却按下文字的鍵），系統就會給您一道警告的訊息，您得要全部重新輸入；APP-LESOFT會顯示REENTER，然後重新執行 INPUT 敍述，方法如前所述，因此您就得全部地重頭來過了。

數值的輸入必須要有正確的數字符號；如果在需要數值資料時，您

只單單按下RETURN鍵，您就會收到一個錯誤訊息，因此就要重新輸入了。正確的數值輸入符號只能是0到9、空格、正號與負號而已；就APPLESOFT來說，實數的輸入還可以有小數點，以及為了指數輸入時的E字和指數那兒的正負號。

在APPLESOFT中，如果字串輸入的第一個非空白的符號是個引號，那麼一直到下一個引號或RETURN的符號（包括了逗號與冒號）就存入需要輸入的字串變數。如果輸入資料並非用引號開始，則到下一個逗號、冒號、或RETURN以前的符號（含引號）會存入該對應的字串變數。若一個INPUT敍述中要求好幾個字串的輸入，則輸入字串之間得要用逗號分開。

當字串變數需要輸入資料時，您只是按下RETURN，則此字串變數就會得到個虛無字串（即“ ”）。

APPLESOFT中，除非輸入資料用引號作為開始，要不然冒號以後的所有資料都會被忽略掉的。

直接式中不可以用INPUT。

INT

讓APPLE Ⅱ進入整數BASIC。

格式：

INT

任何目前在記憶體中的程式都會被洗掉。如果系統中沒有整數BASIC，則會產生LANGUAGE NOT AVAILABLE（無此程式語言）的訊息。

只能用於直接式。

INVERSE

這一道APPLESOFT敍述把螢光幕變成白底黑字。

格式：

INVERSE

自這道敍述執行後，任何 PRINT 所顯示的都是白底黑字，就連錯誤訊息也不例外；不過，却不影響您輸入給 INPUT 的資料；當然，在此敍述執行之前所顯示出來的內容也是不受影響的。

INVERSE 的功能是利用把ASCII碼稍做修飾而來的，所以在白底黑字時所存到磁碟上的文字碼都是不正確的，是故當您把它再讀回來時，就會得到不正確的結果。

整數 BASIC中無此功能。

LET=

這是道設定敍述； LET= 或簡單地 =，表示將一個值存到某個變數中。

格式：

[LET] *var = expr*

這是把 *expr* 所算出來的值儲存到變數 *var* 中。

LIST

把目前在記憶體中程式的全部或一部份顯示出來。 LIST 指令有兩個型式，其中之一是整數 BASIC 與 APPLESOFT 都能使用的，但是另一個就只能在 APPLESOFT 中用了。下面分開說明這兩種不同的型式。

一般格式：

LIST ***line***$_1$ [, ***line***$_2$]

我們可以用這道指令顯示程式中任何部份。如果 LIST 後頭沒寫任何列編號，就會把整個程式都顯示出來；若只寫了 ***line***$_1$ ，那麼（如果有那一列）就只顯示指明的那一列；如果兩個列編號都寫出來了，就會顯示 ***line***$_1$ 與 ***line***$_2$ 之間的每一道敍述。

如果 ***line***$_1$ 不存在，就從比它大的那一列開始；若 ***line***$_2$ 不存在，就顯示到比它小的那一列爲止。

LIST中的列編號不可以用變數或運算式替代。

當您用 LIST 來顯示程式時，系統會自動地把變數與保留字兩邊加上空格，以便容易閱讀；如果這會造成問題的話，您可以用POKE 33，33 這一道指令來消除這些額外的空格（用POKE 33，40 或 TEXT 就會變回有空格的型式）。

程式列的長度當然是有限制的，但是在 LIST 把空格加進去之前就會把這個限制考慮進去了，所以當您鍵入程式的時候可以不必有空白來加長您程式列的長度，不過當您列出程式而造成加入空白之後，這個程式列也許就太長，不那麼容易編修或抄錄了。

Applesoft擴充格式：

LIST $\left\{ \begin{matrix} \textit{line}_1 \ [\{\overset{,}{-}\}] \\ [\textit{line}_1]\{\overset{,}{-}\} \ \textit{line}_2 \end{matrix} \right\}$

在APPLESOFT中，您可以用逗號（，）或者是減號（－）來把兩個列編號分開。

在APPLESOFT中，若您省略了 *line*$_1$ ，那麼就會從程式的開始到指明的敍述顯示出來；若省略了 *line*$_2$ ，就會從指明的敍述一直顯示到程式的終了。

LOAD

從磁碟或卡帶中抄入某個程式。

卡帶格式：

LOAD

從卡帶中，把下一個程式抄入記憶體，並且取代目前記憶體中的程式（如果有的話）；請注意一點，當 LOAD 執行時，卡式機必需要在PLAY 才行，而APPLE Ⅱ是不會告訴您這一點的。當APPLE Ⅱ 開始抄一個程式時會發出"嗶"的響聲，同樣地，抄完了之後也會響出聲，而這第二聲就是告訴您這是把卡式機停下來的時候了。

整數BASIC 中LOAD只能用在直接式。

磁碟格式：

LOAD *filename*[,D*n*][,S*n*][,V*n*]

從磁碟中把叫做 *filename* 的程式抄到記憶體中；如果 LOAD 執行成功了，那麼就會取代記憶體原來的程式。

如果要抄進來的程式是用 APPLESOFT 寫成的，而APPLE Ⅱ目前却在整數 BASIC 的控制之下，於是APPLE Ⅱ就會自動地轉換成合用的語言；這可能會造成有一個從磁碟中抄入適當語言的動作，如果並沒有這套語言，就會顯示LANGUAGE NOT AVAILABLE的訊息。反之亦然。

如果所要抄錄的程式不在S*n*的D*n*中，就會顯示FILE NOT FOUND 的訊息；如果S*n*的D*n*中的磁碟片不是V*n*，那麼VOLUME MISMATCH（磁碟片不對）就會出現。

D*n*、S*n*與V*n*可以用任何次序來寫。如果省略了D*n*或S*n*，於是就使用上一次的值；如果V*n*沒寫，就假定爲V0；另外，*n*也可以不寫，這個時候就會假設爲D0、S0或V0。

這是一道DOS指令，在間接式時得要使用 PRINT 與CTRL-D的技巧。

LOCK

把某個磁碟檔保護起來，以免意外地把它毀掉。

格式：

LOCK *filename*[,D*n*][,S*n*][,V*n*]

當您把一個檔案鎖住之後，除非您把它開鎖（unlock），那麼這個檔案就不可以被刪掉（delete），或者是改名（rename）了；並且也不可以用鎖住了的檔案作命名，來保存另一個檔案。在磁碟目錄（catalog）中，被鎖住檔案在檔案類型的左方會出現一個星號。

如果該檔案不在S*n*的D*n*中，則會發生FILE NOT FOUND（找不到檔案）的錯誤訊息；若S*n*的D*n*中磁碟片不是V*n*，就會顯示VOLUME MISMATCH（磁碟片不對）的訊息。

D*n* 、**S*n*** 與 **V*n*** 可以用任意順序來寫；如果不寫 **D*n*** 或 **S*n***，就用上一次的值替代；若不寫 **V*n***，就假定爲 **V0**。另外，***n*** 也可以省略，這個時候就相當於 **D0**、**S0** 或 **V0**。

這是一道 **DOS** 指令，在間接式時要用 **PRINT** 與 **CTRL-D** 的寫法，

LOMEM:

定出 **BASIC** 程式中的變數等等所能用到記憶體的下限。

格式:

LOMEM: ***exprnm***

LOMEM: 建立出您程式在 **RAM** 中儲存程式列與變數的最低位址。監督程式與 **BASIC** 翻譯程式把比 **LOMEM:** 小的區域留給列表機，低解析度圖像，以及螢光幕區域等等之用。當我們把 **APPLESOFT** 從卡帶或磁碟抄入記憶體時，它永遠儲存在比 **LOMEM:** 還小的區域。您也可以用 **LOMEM:** 敍述多留出一些位置，以便給機器語言副程式、高解析度的造型表（ **SHAE TABLE** ）使用。

在整數 **BASIC** 或 **APPLESOFT** 卡時，**LOMEM:** 從2048算起，這正好在系統區域之後；一旦您從磁碟或卡帶抄入 **APPLESOFT** 之後，**LOMEM:** 就變成12291了。每一次您加入或改變一列 **APPLESOFT** 敍述時，**LOMEM:** 就會隨著增加或減少；同樣，洗掉（用 **NEW** 或 **CTRL-B** ）一個 **APPLESOFT** 程式也會改變 **LOMEM:** 。因此之故，如果您要在您程式之前留出一些空間的話，您得要在抄入或鍵入新程式之前，洗掉任何以前就留在記憶體內的程式之後。

exprnm 的值必須要在－65535到65535（整數 **BASIC** 是

－32767到32767 ）之間，要不然就會導致錯誤訊息出現。

您可以用PRINT PEEK（106）＊256＋PEEK（105 ）來顯示目前 LOMEM：的值。

在APPLESOFT中，如果您把 LOMEM：的值定得高過HIMEM：的值，或者低過現有 LOMEM：的值，或者是因而與目前的作業系統或程式相重疊，就會導致 ?OUT OF MEMORY ERROR （記憶體不夠用）的訊息。

整數 BASIC 中只能在直接式下使用。

MAN

在整數 BASIC 中終止自動編列編號的動作。

格式：

MAN

自動編列編號的功能是用AUTO來起動的。

鍵入C_{TRL}－X暫停產生列編號的動作，然後再鍵入MAN。

APPLESOFT無此功能。

MAXFILES

這一道敍述指出同時使用的最多磁碟檔的數目。

格式：

MAXFILES *limit*

DOS 中您最多只能開啓（同一時刻）十六個檔案。當執行這道敍

述的時候，系統會爲您把每一個檔案在記憶體中留出595個 byte 的區域，叫做檔案緩衝區（ file buffer ）；當您把 DOS 抄入記憶體時，系統會把MAXFILES定成3。

除了MAXFILES之外，每一道DOS指令執行的時候都需要一個檔案緩衝區，所以，實際上您所能用到的檔案緩衝區的數目應該是 MAXFILES 中所定出來的減去一。當您在沒有檔案緩衝區可用時，又執行了DOS指令，於是就會出現 NO BUFFERS AVAILABLE （沒有檔案緩衝區可用）。

當執行MAXFILES時會改變 HIMEM: 的值，因此可能毀掉您的程式或變數等等；所以，如果可能的話，您應該在抄入或執行您程式之前就執行這一道命令。

如果在APPLESOFT程式中一定要用這道MAXFILES敍述，那麼請寫在第一列；在整數 BASIC 時要用MAXFILES，您必須要先產生一個 EXEC 檔才行（見第五章或本章的 EXEC ）。

limit 必須要是1到16之間的整數常數，要不然就會發生 RANGE ERROR （不正確的範圍）錯誤。

這是一道DOS命令，在間接式時要使用 PRINT 與 CTRL-D 的技巧。

MON

造成把磁碟的指令以及（或者）把資料的流程在螢幕上顯示出來。

格式：

MON [C][,I][,O]

這三個參數決定究竟要顯示些什麼。如果寫了C，就表示要把所有磁碟的指令在螢幕上顯示出來；寫了I，表示顯示所有從磁碟輸入的資

料；0表示顯示所有從APPLE Ⅱ輸出到磁碟的資料。這些參數可以用任何順序來寫；不但如此，您還可以任意選擇一個或數個的組合。如果一個都不要，MON就沒有作用。MON的作用一直會延續下去，直到NOMON、FP或INT執行了，或重新抄入系統，或者是某些機器上按下RESET爲止。

這是DOS指令，在間接式時要用 PRINT 與 CTRL－D 的寫法。

NEW

把目前在記憶體中的所有變數與程式刪掉。

格式：

NEW

NEW也會改變 LOMEM: 的值，不過對 HIMEM: 、COLOR、或HCOLOR都沒有影響。

整數 BASIC 中，NEW只能用在直接式。

NEXT

終止一個由FOR敍述所啓動的廻圈。

一般格式：

NEXT *varnm*[, *varnm*……]

當執行NEXT的時候，廻圈的變數 *varnm* 就加上在FOR敍述所指明的值，而程式的執行就依著FOR敍述中的其它值來決定是否回到對應的FOR的下一道敍述，還是繼續往下執行。請參看本章中對

FOR的說明。

如果目前並沒有以 ***varnm*** 做FOR變數的廻圈，就會導到?NEXT WITHOUT FOR ERROR（沒有對應的FOR）的錯誤訊息（APPLESOFT），或者是 ***BAD NEXT ERR（整數BASIC）。

如果NEXT之後有好些個變數，那麼必需要以廻圈起動的相反順序來寫，也就是先寫後啓動的那個廻圈的變數，要不然就會發生錯誤。

整數 BASIC 中，NEXT不可以在直接式中使用；在APPLESOFT 時，一道 NEXT 敍述會回到剛才正在執行但還沒有完成的那個FOR敍述。

Applesoft的另一格式：

NEXT

在APPLESOFT中，NEXT後面可以不寫變擇名稱，事實上，所用的變數就是最後啓動而沒有完成的廻圈的變數。沒有變數的NEXT要比有寫變數的NEXT快得多。

NO DSP

在整數BASIC中取消DSP的動作。

格式:

NO DSP *var*

APPLESOFT中無此功能。

NOMON

中止由MON起動顯示磁碟指令或資料流程的動作。

格式:

NOMON [C][,I][,O]

每一個參數都是中止由MON起動的對應部份。如果寫了C，就不顯示磁碟指令；如果寫了I，就停止顯示由磁碟輸入到APPLE Ⅱ的資料；寫了O，那麼從APPLE Ⅱ寫到磁碟上資料就不顯示出來了；這些參數可以任意組合，也可以用任意順序來寫；如果不寫任意參數，或者是MON沒有作用，或者是沒有對應的參數，NOMON 是沒有作用的。

這是道 DOS 指令，在間接式時要用 PRINT 與 C_{TRL} - D的寫法。

NORMAL

這一道APPLESOFT敍述取消FLASH與 INVERSE的動作。

格式:

NORMAL

請參看FLASH與INVERSE。
整數BASIC無此功能。

NO TRACE

取消由 **TRACE** 啓動對程式追踪的動作。

格式:

NO TRACE

如果 **TRACE** 沒有被啓動，那麼 **NO TRACE** 就沒有作用。

ONERR GOTO

當一個 **APPLESOFT** 程式的錯誤發生時，就轉向到所指明的那一列去。

格式:

ONERR GOTO *line*

這道敍述告訴系統，說是有錯誤發生時，就轉向到指明的那一列去；這道敍述（ **ONERR GOTO** ）必須得在錯誤發生之前執行才有效用。

每一類錯誤都有一個代號；最近發生的那一個錯誤代號就存在記憶體中222那兒，因此您用 **PEEK**（222）就可取得這個代號，這個代號的意義請參看表C－1（在附錄C）。如果在 **FOR-NEXT** 廻圈或副程式中發生了錯誤，這會把系統中的指標（ pointer ）與資料疊（ stack ）弄亂，因此，您程式中處理錯誤的部份必須要回到對應的 **FOR** 或 **GOSUB** 敍述，重新地執行這個廻圈或副程式；如果您處理錯誤的程式並不如此做，而是回到 **NEXT** 或 **RETURN** ，那麼就會發生另

一個錯誤了。

在某些狀況下，ONERR GOTO 不一定能夠正常作業。在 APPLE Ⅱ中，如果同一列有兩個GET的錯誤，而處理錯誤的程式使用 RESUME 而非 GOTO 把控制權轉回時，系統就會鎖住，不會有任何動作。還有，如果您的程式使用 PRINT 敘述（或者是 TRACE 正在有作用），於是第 43 個錯誤若不是由 INPUT 產生的話，就會造成把控制權交到監督程式（Monitor），如果在這個情況下，您用 GOTO（而非 RESUME）來終止錯誤處理程式的動作，那麼第 87 次 INPUT 的錯誤還是會進入監督程式的。

這些困難都是可以克服的，解決之道在於您得在錯誤處理程式中加上一道叫用表 8－4 中所列的機器語言副程式就行了。

表 8－4 ONERR GOTO的機器語言修整程式

機器語言		6502 組合語言	
十進位	十六進位	指令	註解
104	68	PLA	將堆疊第一個 byte 放入加法器
168	A8	TAY	保留到Y暫存器中
104	68	PLA	將下一個 byte 放入加法器
166	A6	LDX $DF	以ONERR指標
223	DF		
154	9A	TXS	做為堆疊位址
72	48	PHA	將已保存的堆疊內容放入
152	98	TYA	'ONERR' 堆疊內（加法
72	48	PHA	器兩個 byte、Y暫存器）
96	60	RTS	回到APPLESOFT

您可以使用 POKE 敍述把上面的十進位值存入記憶體 768 到777（或者是任意您可以用的區域）中，然後在錯誤處理程式中CALL 768就行了。

整數 BASIC 無此功能。

直接式時不可以使用。

ON-GOSUB

在 APPLESOFT 中有條件地叫用好幾個副程式中的一個，這個選擇由某個運算式目前的值來決定。

格式：

ON *exprnm* GOSUB *line* [, *line*……]

如果運算式的整數值爲 1 就叫用在第一個列編號的副程式，爲 2 就叫用在第二個列編號的副程式，餘此類推。一旦在該副程式中執行到 RETURN 敍述，就把控制權轉回到ON-GOSUB的下一道敍述。

運算式的值必須要在 0 到 255 之間，不然就會出現 ? ILLEGAL QUANTITY ERROR（不合法的值）的錯誤；若運算式的值爲零，或大於後面所列出的列編號數目，那麼這道敍述就沒有作用，系統就執行 ON-GOSUB 的下一道敍述。

整數 BASIC 中無此功能；不過，您可以參看整數 BASIC 中計算式 GOSUB 的一個特殊型式。

ON-GOTO

在 APPLESOFT 程式中，利用運算式的值有條件地轉向到好幾個列編號的其中一個。

格式：

ON ***exprnm*** **GOTO** ***line*** [, ***line*** ……]

如果運算式的整數值爲 1 ，就轉向到後面的第一道列編號，爲 2 就是第二道列編號，等等。

運算式的值得在 0 到 255 之間，不然就會出現 ? ILLEGAL QUANTITY ERROR （不合法的值）的錯誤；如果運算式的值爲 0 或者是大過後面列編號的個數，那麼程式就執行ON-GOTO的下一道敍述。

整數 BASIC 無此功能，不過您可以參考整數 BASIC 中的計算式 GOTO 的型式。

OPEN

準備好一個順序文字檔或者是直接式磁碟文字檔，以備後頭使用。

格式：

OPEN ***filename*** [, **L*n***] [, **D*n***] [, **S*n***] [, **V*n***]

首先，系統爲這個文字檔案留出一個 595 個 byte 長的檔案緩衝區，於是我們就可以用 READ 或 WRITE 來取出或儲存資訊了；若指明的檔案並不在磁碟中，系統就會爲您產生一個；如果該檔已經被開檔（ open ）了，系統會先把它結檔（ close ），再把它打開。

如果沒寫L參數，那麼這個檔案就以順序檔來開檔。

如果寫了L參數，於是這個檔案就用直接檔來開檔，L後面的數目就是這個檔案以 byte 爲單位的錄（ record ）長度，它得要是 1 到 32767 之間的整數常數，如果不寫這個常數，系統就當做L 1 來處理。請注意，每一次把同一個直接檔開檔時，這個錄長度必須要相同。

如果 S*n* 的 D*n* 中磁碟片不是 V*n*，就會造成VOLUME MIS-MATCH（磁碟片不對）的錯誤。

D*n*、S*n* 與 V*n* 可以用任意順序來寫；如果沒寫D*n* 或 S*n*，就用上一次使用的值替代；如果省略了V*n*，就設定爲V0；另外，*n* 是可以不寫的，這個時候系統會用D0、S0與V0來替代。

這是道DOS指令，在間接式時要用 PRINT 與 C_{TRL}-D 的寫法。

OPEN 不可以在直接式中使用。

PDL

在本章有關函數的部份討論。

PEEK

在本章有關函數的部份討論。

PLOT

在低解析度螢幕上描出一個點。

格式：

PLOT *col*, *row*

在低解析度繪圖方式之下，PLOT 敍述在螢幕上描出一個有顏色的點，它的顏色由上一次執行的 COLOR 敍述所決定。行的值要在 0 與 39 之間，第 0 行是螢幕的最左邊邊緣，第 39 行是最右邊的邊緣；列的值要在 0 與 47 之間，第 0 列是螢幕最頂上的一列，第 47 列是最

低下的一列。在第 40 到 47 列上的點會出現在文字幕（ text window ）中，不過您可以用POKE - 16302，0 來消出這四列文字幕。

在文字式或者是文字幕中，PLOT 不是描出一個點，而是顯示一個字；因爲一個字要用到垂直的兩個點，因此就有兩個不同的座標值在 PLOT 中造成顯示同一個字的現像。是故，若您要把某個特殊的字在螢幕上顯示出來，您必須要把兩半都用 PLOT 點出來才行；至於究竟出現的是那一個字母，那就得看點出這兩半以前所執行的 COLOR 敍述了。

表 8 － 5 告訴我們不同顏色的組合所產生文字的上半與下半。要產生一個特殊的文字，您先在表 8 － 5 中找到那一個字母，然後找出在表左邊與在表上方的顏色代號（上、下半都得找出來）；字的上半部就由 PLOT 偶數列，下半部就用一個 PLOT 奇數列的方式顯示出來，但是若您只點出一個文字一半的位置，那麼所產生出來的字母就由PLOT所用的顏色，以及在其交點所用的顏色來決定。

表 8 － 5　文字式中圖型敍述所產生的文字

	螢幕的狀態				上半部圖型的顏色															
	反白字	閃爍字	正常字		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
下半部圖型的顏色	0	4	8	12	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	1	5	9	13	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
	2	6	10	14	■	!	"	*	$	%	&	'	(	)	•	+	,	-	.	/
	3	7	11	15	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?

請注意，在低解析度時，每一個文字都可以由四種不同的顏色產生出來，不過，都不是一樣的；也就是說，當下半部顏色小於 4 時，就是反白字，顏色在 4 與 7 之間爲閃標字，若在 8到15之間就是正常的字了

POKE

POKE 敍述把一個BYTE 的資料儲存到某個記憶體位置中。

格式：

POKE *memadr, byte*

一個由 byte 所提供，在 0 與 255 之間的值，會被存入記憶體中 *memadr* 那兒；如果這個記憶體的位置超過了您擁有的記憶體（比如說，若您有 16K 就是 16383），或者是使用到一個不能接受資料的輸出裝置，那麼POKE是沒有作用的。

您必須要小心使用POKE ，因爲記憶體中有某些位置的內容是APPLE Ⅱ 爲了使整個系統運轉而獨有的，因此不能被毀掉，所以如果不小心使用就會破壞您的記憶體內容，您的程式，甚至於連BASIC編譯程式或系統都破壞掉。

POP

使APPLE Ⅱ忘掉最近一次執行GOSUB 時所必須返回的位置。

格式：

POP

POP的作用就等於是把最近執行的那個GOSUB 敍述轉換成爲GOTO敍述，因此一旦您執行到一個RETURN 時，那末轉回去的所在就是目前往回數的倒數第二次GOSUB 的下一道敍述了。如果程式中執行POP與RETURN的總數超過了所有GOSUB 執行的總數，

就會發生一個錯誤的訊息。

POSITION

把磁碟檔指標自目前的位置向前特定個位置。

格式:

POSITION ***filename*** [,***Rn***]

當執行POSITION 時，若該檔還沒有打開，那麼系統就會先開檔（見OPEN ）；R後面的數目指明要從目前位置向前移的欄位數，這個數必需要在 0 與 32767之間，如果不寫它，這個值就假定爲零—也就是說不移動檔案指標。如果這個檔案是被POSITION 開啓的，那末欄位就從檔案的開端起點，這個檔案必須要是順序檔。

所謂的欄位，就是一串文字最後跟著一個回頭鍵符號；POSITION 敍述會把一個檔案從目前的位置向前查過去，一個文字一個文字地檢查，計算回頭鍵符號出現的個數，因此回頭鍵出現的次數就是跳掉欄位的數目。不過，若是所要跳過的欄位還沒有足夠，但就遇到了檔案中未曾使用的部份，就會給出END OF DATA 的訊息。

這是一道DOS指令，在間接式中要用到PRINT與C_{TRL}-D 的技巧來寫。

立即式時不能使用POSITION。

PR#

選擇一個要接受未來輸出的週邊裝置接點。

格式:

PR# *slot*

當執行過這道敍述後，所有的PRINT 輸出都會送到指明的那一個接點上的週邊裝置去；*slot* 的值必須要在 0 與 7 之間。請注意，第 0 號接點並非週邊裝置，而 PR# 0 的意義就是選擇 40行的螢光幕做爲輸出機具。如果指定的接點上沒有週邊機具，系統就會鎖住不會有所動作，直到您按下RESET爲止。

如果記憶體中存有DOS，那末PR# 就會當成DOS 指令來處理，因此在間接式時就需要PRINT與CTRL－D的寫法了。

PRINT

把文字在螢光幕或其它輸出機具上顯示出來。

格式:

PRINT [***expr***] [{ ; , } ...... [***expr***]] ......]

PRINT敍述的寫法有好幾個變化。如果只寫了PRINT ，那就是送出一個回頭動作並且跳到下一列開端；若PRINT 中有幾道運算式，則會印出這些運算式的值，這些值顯示的方式就由分離這些運算式的逗號或分號來決定。

整數BASIC所有的值以及APPLESOFT 的很多值都用標準數值表示顯示，負數前面有一個負號，但正數前面却不會有一個正號或是空白。在APPLESOFT中，比±.01 還小的值，或是小數點前面多過九位數的值就用科學記法顯示。

字串值就依它們的內容直接顯示。

逗號與分號可以決定兩個值之間的空格數。在分號的情形，會使得下一個要顯示的數會緊緊地跟在上一個已經顯示出來的數後面，兩者之

間沒有空格。但逗號却會使得下一個要顯示的數與目前剛被顯示出來的數之間留有幾個空格，在下一個定位的所在顯示出來。

整數BASIC的定位是每八個格子一組，這就是1，9，17 ……等等，但是如果有一個非空白的符號印在某一個定位上時，這個定位就不發生作用了。

APPLESOFT 中却是16格一組的，這就是1，17，33，……等等，請參看第四章的圖4－1，那兒說明了什麼時候一個定位不會發生作用；但是，對其它的輸出機具而言，却是在某個定位之前一個格子（比如說16，32，……）有一個非空白的符號時，這個定格就不發生作用了。

如果在最後一個運算式後面並沒有跟著一個逗號或分號，那末當把所有值都顯示完後就會送出一個回頭動作，接著就換行；如果最後是一個分號，於是下一道PRINT 敍述的第一個資料就會接在後面顯示出來，而且中間沒有空白；若最後是一個逗號，那末下一次輸出就在下一個定位位置顯示出來。

在整數BASIC 時，所有項目之間必須要有逗號或分號，而APPLESOFT 中却可以不必，於是這些項目的輸出就像是用分號一般地接起來了。

APPLESOFT 中可以用問號（？）來代替PRINT 這一個字，不過在顯示程式時，您却會見到PRINT這個字的。

READ（磁碟敍述）

定出一個可供以後INPUT與GET敍述使用的磁碟檔。

格式：

READ ***filename*** [,***Rn***] [,***Bn***]

如果這個檔案尚未開檔，那末就會先把它開檔（OPEN ），於是

所有的INPUT與GET敍述都從磁碟上接收文字輸入，一直到另一個磁碟敍述（或CTRL－D）被印出，或按了RESET，或有錯誤出現時爲止。如果這個檔案並不在磁碟上，就會出現FILE NOT FOUND（找不到檔案）的訊息。

若不寫R的部份，這個檔案就當成順序檔來讀了；B指出READ敍述從第幾個 byte 開始讀資料，但若省略了B，則從這個檔案的第一個 byte （即 byte 0 ）開始讀。如果一個 byte 的內容並沒有經由 WRITE 指令把資料寫上去，那麼在以後的 INPUT 或GET 執行就會顯示出END OF DATA（資料終了）的訊息。

如果寫了R，這個檔就看成直接檔，或是隨機處理檔來看了，n就指出READ要使用的是第 n 個記錄。

在隨機處理檔中，B部份指出的就是從這個記錄中第 n 個 byte 開始讀取資料；如果沒寫B，就會從記錄的第一個 byte （即 byte 0 ）讀起。但若讀入的 byte 先前並沒有經由 WRITE 指令寫入資料，那麼在以後的 INPUT 或GET 敍述執行時就會出現END OF DATA（資料終了）的訊息。

跟在B與R後面的必須要是一個0 到 32767之間的整數常數，如果不寫就假定爲零。

APPLESOFT中千萬不要用CTRL－C來停止READ 的動作，因爲這會導致一連串 ?REENTER 的訊息出現；如果要這樣做，那末就請使用RESET來停止程式的動作。

這是個DOS 指令，在間接式時要用PRINT與CTRL－D的方式來寫；在直接式（或立即式）時不一定可以使用。

READ（設定敍述）

在APPLESOFT中，從DATA中找出資料給變數。

格式：

READ ***var*** [, ***var*** ……]

系統中有一個指標，指向DATA敍述中可以給RFAD 中第一個變數的值。在程式一開始執行，以及執行完RESTORE 敍述之後，這個指標就會指向DATA的第一個值；接著，每一個READ讀取一個值之後，指標就指向下一個值。

變數可以是任意的型別，不過必須要跟DATA 中對應的資料相吻合；數值資料給字串變數不會出什麼問題，但字串資料給數值變數就會造成？SYNTAX ERROR 這道訊息出現了。出問題的DATA 敍述的編號也會一起被顯現出來。

如果 READ 的資料比 DATA 中還多，就會顯示？OUT OF DATA ERROR 的訊息，出問題的 READ 敍述編號也會被顯現出來。

只要程式中 DATA 敍述含有足夠的資料，直接式中也是可以執行 READ 敍述的，要不然依然會出現？OUT OF DATA ERROR 的訊息。如果記憶體中有 DOS 存在，則在直接式中執行 READ ，那麼這個 READ 就會被當做 DOS 指令來看，於是您就會得到 NOT DIRECT COMMAND（並非直接式命令）的訊息。

整數BASIC 無此功能。

RECALL

在 APPLESOFT 中，自磁帶上取出一個數值矩陣的值。

格式：

RECALL ***varnm***

APPLESOFT 會停下來，一直等到在磁帶上找到了那個矩陣爲止，在這段期間中沒有其它的指令可以執行； RECALL 本身並不控制磁帶的轉動，而且也不會提醒您何時可以讓磁帶機進入 PLAY 的方式，因此您在執行 RECALL 之前應該有 PRINT 敍述提示使用者，在執行完之後也是一樣告訴使用者停止磁帶機轉動。當開始讀取矩陣的資料時，APPLE Ⅱ會發出“嗶”的一聲，讀完了之後也會再“嗶”一聲；請注意，這個矩陣必須在使用 RECALL 之前就先宣告它的容量，要不然就會出現 ?OUT OF NEMORY ERROR 的訊息（見DIM）。

在 RECALL 與 STORE 中的變數名稱不一定要相同，不過讀回資料時所用的變數維數與容量却必需要與寫出去的時候一致；若存起來時的元素比較多，就會出現 ?OUT OF MEMORY ERROR；若讀入時矩陣元素個數至少有存入時的個數，但是維數可能不同，那麼就會出現 ERR 的訊息，但是程式却是會繼續執行的。

如果在讀回來時的矩陣元素較多，那麼讀完之後矩陣中的元素可能會被擠成一塊了。這有兩個例外；第一，如果您要讀回資料的矩陣維數與寫出資料時矩陣維數完全相同，而且除了最後一個維數之外，每一個維數的容量也完全一樣，而最後一維的容量可能大一點，那麼矩陣資料是可以被讀回來的，最後一維多出來的那些元素就都存了零。第二，在記憶體內陣列元素的個數至少有在磁帶上將要被讀回來的陣列元素那麼多，這個時候會給出一個 ERR 訊息，不過程式會繼續執行。

在 RECALL 中不能用字串陣列，不過要讀回來的數值您却可以用 CHR$ 函數轉換成字串。

整數BASIC 中無此項功能。

REM

這道註解敍述可以讓我們在程式中加入一些對程式的說明及註解。

格式：

REM ***comment***

此地***comment***爲任意的，可以寫到一個程式列中的文字。

在列出程式時，註解是一併列出來的，不過在執行時卻不會有任何動作；**REM**可以自己獨佔一列，也可以是一列中好多敍述的最後一個。要注意的是，如果一列中有好些個敍述，則**REM**應該是最後一個，因爲在這個字之後的所有內容都當成註解來看。

RENAME

在不改變檔案內容的情況下把檔案改名。

格式：

RENAME ***filename***$_1$**,** ***filename***$_2$ [**,Dn**] [**,Sn**] [**,Vn**]

在磁碟片上找出***filename***$_1$這個檔案，把它的名稱更改成***filename***$_2$，如果這個檔案還是開啓的，那麼就把它關閉（見**CLOSE**），除此之外就沒有會影響到這個檔案的所在了。

如果磁碟中有要被改名的檔案，那麼改名的動作是必定會成功的，它並不會檢查要改的名字是否已經重覆了，所以在**RENAME**執行之前，您得要確定磁碟中果然沒有叫做***filename***$_2$的檔案。

如果在**Sn**的**Dn**中沒有***filename***$_1$就會顯示**FILE NOT FOUND**的錯誤訊息；若**Sn**的**Dn**中的磁碟片不是**Vn**，就會給出**VOLUME MISMATCH**。

Dn、**Sn**、**Vn**可以用任何順序來寫；如果**Dn**或**Sn**沒寫，就用上一次的磁碟機或接點來替代；若省了**Vn**，就假定爲**V0**；又***n***也

可以不寫的，這個時候就相當於 D0 、S0 或 V0 了。

這是個 DOS 指令，在間接式時需要 PRINT 與 C_{TRL} - D 的寫法。

RESTORE

把 APPLESOFT 敍述中 DATA 的指標移到第一個資料。

格式：

RESTORE

以後的 READ 敍述再讀資料的話就讀取所有 DATA 敍述中的第一個資料。

在整數BASIC中無此功能。

RESUME

當一個錯誤發生之後，讓 APPLESOFT 的程式重新從發生錯誤的指令開始執行。

格式：

RESUME

RESUME 只能用在當錯誤發生因而造成ONERR GOTO 敍述發生作用之後；如果沒有錯誤發生，但却執行了 RESUME 那結果將是不可預測的，不過通常却會叫您悲哀就是了。

整數 BASIC 無此功能。

立即式中不可以使用。

RETURN

讓程式返回到最近執行的 GOSUB 執行的下一道敍述。

格式：

　　RETURN

POP 敍述會毀掉最近執行的 GOSUB 敍述的內容，所以在執行過 POP 之後再執行 RETURN ，那麼就會返回到最近倒數第二次所執行的 GOSUB 之下一敍述。

如果 RETURN （或 POP ）的執行次數多於 GOSUB 執行的次數，那麼就會出現錯誤訊息。

ROT=

這一道 APPLESOFT 敍述設定由 DRAW 或 XDRAW 在高解析度圖型下的旋轉方向。

格式：

　　ROT= *exprnm*

ROT=0 把造型依所定義的方向畫出來。 *exprnm* 的值每增加 16 ，造型就沿順時針方向轉 90 度；因此，ROT=32 等於上下倒置，ROT=64 就是原來的方向。如果 *exprnm* 的值大於 64 ，就以64去除後所得的餘數爲準。

如果 SCALE 定成1，那麼 ROT= 的值就只認得四個，即0、16、32、與48； SCALE=2 時就有8個值； SCALE=3 就

有 16 個值等，一直到最多 64 個值爲止。

exprnm 的值必須要在 0 到 255 之間，要不然當 ROT= 執行時就會出現 ?ILLEGAL QUANTITY ERROR 的訊息。

如果 ROT 後頭不跟著等號，就不會被當做保留字看了。

整數 BASIC 無此功能。

RUN（磁碟敍述）

從磁碟中抄入一個程式，並且馬上執行它。

格式：

RUN *filename* [, D*n*] [, S*n*] [, V*n*]

系統會把磁碟中叫做 *filename* 的檔案抄入記憶體，然後執行，如果抄錄的動作成功了，那麼原來在記憶體內的程式就會被洗掉。

如果要抄錄的程式是用整數 BASIC 來寫的，而目前 APPLE Ⅱ 內却是 APPLESOFT（反之亦然），那麼機器會先轉變到適當的語言去，如果有必要的話還會從指定的磁碟中先把 APPLESOFT 編譯程式抄進記憶體。如果系統中無此語言，就會顯示出 LANGUAGE NOT AVAILABLE 的訊息。

D*n*、S*n* 與 V*n* 可以用任何順序來寫；如果省略了 D*n* 或 S*n*，就用上一次的值替代；如果不寫 V*n*，系統會認爲是 V0；如果 *n* 沒寫，就假定成 D0、S0 與 V0。

這是個 DOS 指令，在間接式時要用 PRINT 與 CTRL－D 的寫法。

RUN（一般的敍述）

把目前在記憶體中的程式自指定的列編號（如果有的話）起開始執行，如果沒寫列編號，就從號碼最低的那一列開始。

一般格式：

RUN[*line*]

如果程式中沒有 *line* 這個號碼，您就得得到 ***BAD BRANCH ERR的訊息（整數BASIC），或者是?UNDEF'D STATEMENT ERROR（在 APPLESOFT）。

整數BASIC中另一格式

RUN *exprnm*

在整數BASIC中，列編號可以是一個運算式。
在整數BASIC中，RUN只能在直接式下執行。

SAVE

把目前在記憶體中的程式保存到卡式帶或磁碟上。

磁碟格式：

SAVE

這會把目前記憶體中的程式儲存到卡式帶中，當 SAVE 執行時，您的卡帶機必需要在 RECORD 的方式，因爲APPLE Ⅱ並不會告訴您要這樣做。在開始抄錄程式時，APPLE Ⅱ會"嗶"地一聲，而1抄錄完成後也會發聲，當您聽到第二聲"嗶"的時候，您就應該停止卡帶機的動作了。

在整數BASIC時只能在直接式（或叫做立即式）時使用。

磁碟格式

SAVE *filename* [, D*n*] [, S*n*] [, V*n*]

如果目前磁碟中並沒有叫做 *filename* 的檔案，那麼APPLE Ⅱ會先以目前您所用的語言產生一個對應格式的檔案，然後再存入現在儲存在記憶體內的程式；如果磁碟片已經有了一個叫做 *filename* 的檔案，那麼就會先洗掉原來的檔案才把目前的程式儲存起來。但是，若有 *filename* 這個檔案，不過所用的語言，或者檔案型態都不一樣，您就會得到一道FILE TYPE MISMATCH的訊息。

若接點 S*n* 上的磁碟 D*n* 的磁碟片號碼不是V*n*，就會導致VOLUME MISMATCH的錯誤訊息。

D*n*、S*n* 或V*n* 可以依任何順序來寫；如果省了D*n* 或 S*n*，就用上一次使用時的值代替；如果不寫V*n* 就假設為V0；另外，*n* 也可以不寫，這個時候就假定為D0、S0或V0。

這是個 DOS 指令，在間接式時得用 PRINT、CTRL-D 的寫法。

SCALE

這是道 APPLESOFT 的敍述，功用在設定用 DRAW 與 XDRAW在高解析度圖型下繪出造型時的比例大小。

格式：

SCALE = *exprnm*

造型表中的造型全部乘上 *exprnm* 的整數倍數，所以當SCALE

= 1時，繪出的造型就與原來訂出的相同，當 SCALE = 2 時，造型就放大了一倍，……等等。如果 SCALE = 0 ，這卻是原來造型的 255 倍大。

Exprnm 的值得在 0 與 255之間，要不然，當執行到 SCALE 時就會給出一道 ?ILLEGAL QUANTITY ERROR 的訊息。

請注意，如果 SCALE 後面沒有一個等號時，它是不會當做保留字看的。

整數BASIC中無此功能。

SHLOAD

這一道 APPLESOFT 敍述，它從卡帶機上讀取一個高解析度的造型表。

格式：

　　SHLOAD

在第六章中講述了如何構作以及儲存造型表。

當把造型表抄入記憶體時，它是放在 HIMEM: 之下，而且 HIMEM: 馬上就定到造型表的下方，然後您可以在位置 22 與 23 那兒找到造型表在記憶體的位置。

在執行 SHLOAD 之前，您必須要確定您已經把 HIMEM: 定得不致於蓋過程式或變數，而且也不會毀掉了您的圖型。請參考本章中有關 HIMEM: 的說明，以及附錄 G 中的記憶體分配圖；第六章中有更爲仔細的說明。

整數BASIC中無此功能。

SPEED

這一道 APPLESOFT 的敘述改變文字輸出的速率。

格式：

SPEED ***exprnm***

exprnm 的值就用來建立在螢光幕上或其它輸出機具上顯示文字的速率；最慢的是 0 ，最快的是 255 。

整數BASIC中無此功能。

STOP

停止一個APPLESOFT程式的執行動作。

格式：

STOP

APPLE Ⅱ 回到立即式，然後，**BREAK IN** ***line*** 這道訊息就出現了，此地 ***line*** 就是執行 **STOP** 那一列的號碼。

整數BASIC中無此功能。

STORE

在APPLESOFT中把某個指定的矩陣儲存到磁帶上。

格式：

STORE ***varnm***

STORE 不會控制磁帶的運轉，而且也不會告訴您何時要把磁帶機變成 **RECORD** 的狀態；因此當 **STORE** 被執行的時候，您必須要讓磁帶機在錄音的狀態下轉動。所以，您在程式中就得先顯示出告示（用 **PRINT** 敍述），當**APPLE Ⅱ**開始錄資料以及錄完之後都會發出“嗶”的一聲。

您只能用 **STORE** 來儲存數值陣列；至於字串陣列您就得先用**ASC**函數轉換成整數值才可以儲存（見**RECALL**）。

整數**BASIC**中無此功能。

TAB

這道整數 **BASIC** 敍述把游標移到目前這一列中指明的那一行。

格式：

TAB ***col***

執行這道敍述時，游標就在目前的這一列上向左或向右地移到***col***這一行，在移動時不會洗掉已經顯示出來的文字。行號的編法是從1編到40（左到右）。

在**APPLESOFT** 中請用 **HTAB** 敍述；也請參考本章中後面所列的**TAB**函數。

TEXT

把螢幕從任何圖型式回到整個螢幕顯示文字的方式。

格式：

TEXT

系統提示號與游標移到螢幕最後一列；如果您在文字顯示式時執行這個敍述，這就是唯一的結果。如果您用任何方式設定了不是整個螢幕的文字幕，TEXT就把它變回整個螢幕。

TEXT 不會把螢幕洗掉，換言之，它不會把低解析度螢幕記憶體第一頁清除。因為一般文字顯示與低解析度圖型使用同樣的記憶體位置，所以在低解析度圖型方式之下執行 TEXT 會造成螢幕的頭二十列充滿了奇奇怪怪的符號。

TRACE

當每一道敍述執行時，都把它的編號顯示出來。

格式：

TRACE

這是一個偵錯敍述，會造成顯示各個被執行的敍述的編號的功能；不過，它的顯示會與程式的輸出混在一塊兒，以致於很不容易閱讀。您只能用NO TRACE來停止這個動作。

UNLOCK

把磁碟中某個檔的鎖打開，使它可以改變或者把它刪除。

格式：

UNLOCK ***filename*** [, D*n*] [, S*n*] [, V*n*]

如果指明的檔案已經被鎖住了，那麼就把這個鎖去掉；若它根本沒有上鎖，則這道敍述就等於沒有動作。

如果接點 S*n* 上的 D*n* 中沒有這個檔，就會給出 FILE NOT FOUND 的訊息；如果 S*n* 上 D*n* 中的磁碟片號碼不是 V*n*，就會有 VOLUME MISMATCH 的訊息。

D*n*、S*n* 與 V*n* 可以用任何順序來寫；如果不寫 D*n* 或 S*n*，就用上一次的值取代；若 V*n* 不寫，就假定為 V0；另外，*n* 也可以不寫，這時就當做 D0、S0 與 V0 看待。

這是一道 DOS 指令，在間接式中要用 PRINT 與 CTRL-D 的寫法。

USR

請參考本章就函數方面的說明。

VERIFY

檢查磁碟中的某個檔案是否有問題。

格式：

VERIFY *filename* [, D*n*] [, S*n*] [, V*n*]

當一個檔案經由 SAVE、BSAVE 或 PRINT 等指令保留到磁碟上時，系統會為磁碟上這個檔案所在的每一個區（*sector*）都計算一個檢查碼（叫做 *check sum*）。VERIFY 敍述就是把這個指明檔案的檢查碼再計算一次，並且拿它與原有的檢查碼相比較，如果它們相同，就表示這個檔案可能沒有問題，因此不會顯示任何訊息。如果至少有一個檢查碼與新計算出來的相異，就會顯示 I/O ERROR 的

訊息。任何的檔案都可以拿來檢查。

若該檔案不在 S*n* 的D*n* 中，就會顯示FILE NOT FOUND 的訊息；若 S*n* 的D*n* 之磁碟片號碼不是V*n* ，就會給出VOLUME MISMATCH的訊息。

D*n* 、S*n* 、V*n* 可以用任何順序來寫；如果省略了D*n* 或 S*n* ，就依上一次的值爲準；不寫V*n* 時就假定爲V0 ；另外，*n* 也可以省略，此時就用D0 、S0 或V0 替代。

這是 DOS 指令，在間接式時得用 PRINT 與 C_{TRL}-D 的寫法

VLIN

在低解析度時，在螢光幕上畫出一條垂直綫。

格式：

VLIN *row*$_1$, *row*$_2$ AT *col*

在 *col* 上，自第 *row*$_1$ 列到 *row*$_2$ 列畫一條綫段，所用的顏色由上一道執行的 COLOR 敍述決定。如果螢幕現在是文字式，或者是有文字幕存在，則若列號大過 39 時，這條綫段的某部或全部就不再是點，而是以文字出現了。這些文字符號是由上一次所執行的 COLOR 敍述所決定的，見本章的表 8 － 5 以及有關 PLOT 敍述的說明。

在整數 BASIC 中，*row*$_1$的值必須要少於或等 *row*$_2$，要不然就會出現 ***RANGE ERR的訊息。

VTAB

把游標在目前所在的行上向上或向下移到指明的列上。

格式：

VLIN ***row***

游標會在目前所在的行上向上，或向下移，移到指定的列爲止，所經過的符號都不會消失。列數要在 1 與 24 之間（由上往下數）。

WAIT

把 **APPLESOFT** 程式的動作停下來，一直到某個特定的記憶體位置中的值到達某個特定值爲止。

格式：

WAIT ***memadr***, ***exprnm***$_1$ [, ***exprnm***$_2$]

WAIT 敍述檢查在 ***memadr*** 那兒的八個數元（ bit ）或它們的一部份，拿它們與 ***exprnm***$_2$ 的二進位値比較，至於要比較那些個 bit，就由 ***exprnm***$_1$ 來決定。

比較的規則是這樣的：在 ***exprnm***$_1$ 的二進位中，如果某個 bit 爲 1，那麼就把記憶體 ***memadr*** 那兒位置對應的 bit 與 ***exprnm***$_2$ 目標位置的相比；這些在 ***exprnm***$_1$ 中爲 1 的 bit 就叫做指示數元。當所有指示數元都不相等時，就繼續等待；一直到某一剎那有某對指示數元相同了，就會停止等待，而**APPLESOFT**程式就繼續未完成的工作。

如果沒寫 ***exprnm***$_2$，就以零替代。

WAIT 的動作只能被 **R**ESET 或關機打斷。兩個算術式子的值要在 0 與 255之間，要不然就會出現 ?**ILLEGAL QUANTITY ERROR** 的訊息。如果您指明的記憶體位置大過您所擁有的位置（好比說，若您只有 32K，那您就不能超過 32767），或者是使用到一個不能接受資料的輸出機具，那麼 **WAIT** 就會把系統鎖住，除非您按下

RESET。

整數BASIC中無此功能。

WRITE

指出一個以後的多個 PRINT 要送出輸出結果的磁碟檔。

格式：

WRITE ***filename*** [, **R*n***] [, **B*n***]

若指明的檔還沒有開檔（見 OPEN 敍述），那就先把它開檔，在此以後的 PRINT 敍述的輸出就全部送到這個檔內，一直到您只印出一個CTRL－D符號（ASCII 碼爲 4 ）爲止。

如果不寫R的部份，這就當做是順序檔。在順序檔時，B就指出 WRITE 將從那一個 byte 開始存資料，若不寫B部份， WRITE 就從檔案的第一個 byte （即 byte 0 ）放起。

如果寫了R的部份，這個檔案就是個直接檔，寫出的資料就是R後面所指明的記錄。

在直接檔時，B指出在指明的記錄中， WRITE 動作從那一個 byte 起存資料；若省略了B部份， WRITE 就從該記錄的第一個 byte （即 byte 0 ）開始。

B後面的資料也可以用來指出要在既有檔案（或記錄）之前寫出資料；這些資料也可以 READ ，但却不能去 READ 這中間沒有資料的 byte ，要不然就會得到OUT OF DATA的訊息。

R與B後面必須是自 0 到 32767之間的整數，如果不寫就以 0 代替。

當 WRITE 有作用時，APPLE Ⅱ 中原先會送到螢光幕上的文字就都會被送到磁碟上去了，這包含了由 INPUT 所產生的問號以及錯誤訊息等等。

這是一道 DOS 敍述，在間接式時要使用 PRINT 與 C_{TRL} - D 的寫法。

可能無法在直接式（或立即式）之下使用。

XDRAW

這一道 APPLESOFT 的敍述在螢幕上以高解析度方式繪出一個造型，如果第二次再用它，而且給它的資料與上一次完全相同，那麼就會把才繪出的造物洗掉。

格式：

XDRAW ***exprnm*** **[AT** ***colh, rowh*** **]**

這一道敍述繪出造型表中第 *exprnm* 個造型，它每一個點所用的顏色正好是螢幕上那個點的補色；比如說，0 與 3、1 與 2、4 與 7、5 與 6 都互爲補色（見表 8 － 3 ）。在執行 XDRAW 之前，比例與旋轉的參數要用 SCALE 與 ROT 定出。

用 XDRAW 而不用 DRAW 的原因在於您能夠很容易地把一個已經繪好的造型在螢光幕上洗掉，道理在於 XDRAW 是用補色來繪圖的，所以您用同樣的資料把 XDRAW 執行了兩次（或四次、六次……），那麼不管螢光幕上原來有些什麼都不會改變。

如果您在 XDRAW 中沒寫出位置，那麼這個造型就從上一次執行 DRAW、XDRAW 或 HPLOT 的那個點畫起；如果標出了位置，就從那個位置（ *colh, rowh* ）畫起。

造型的號碼必須要大於或等於 0 ，小於或等於造型表中造型的總數（不可以超過 255 ）。

整數 BASIC 中無此功能。

函　數

以下我們討論APPLE Ⅱ BASIC 的函數（以英文字母順序排列），其它的符號意義、簡寫完全遵照本章前面的規則。

很多函數只能在 APPLESOFT 中使用，我們也一併地指明出來。

ABS

傳回某個數的絕對值，這就是把那個數的符號拿掉不管就行了。

格式：

ABS (*exprnm*)

ASC

傳回某個文字的ASCII碼。

格式：

ASC (*expr $*)

如果字串有多於一個文字，則 ASC 只會傳回這個字串的第一個文字的ASCII碼，傳送回來的並不一定是該文字最低的ASCII碼（即 0 到 95 ）。從 96 到 255 的ASCII碼所產生的文字不過是前面文字的重覆而已，不過，在比較（即<、>、=）時卻不當做相同文字看待；在列表機或者是其它輸出機具上可能會把它們當做不一樣的符號來處理。若 *expr $* 的第一個符號是$C_{TRL}$ - @ （ ASCII 碼爲 0 ），則會

顯示出 ? **SYNTAX ERROR** 的訊息；若 ***expr* $** 爲虛字串，就會導致 ? **ILLEGAL QUANTITY ERROR** 的訊息。

在附錄 I 中列出了所有 **ASCII** 字碼。

ATN

傳回引數中的反正切函數。

格式：

ATN (*exprnm*)

計算 ***exprnm*** 的反正切函數值，結果爲弧度，並且在 $-\frac{\pi}{2}$ 到 $\frac{\pi}{2}$ 之間。

整數 **BASIC** 中無此功能。

CHR $

傳回某個 **ASCII** 碼所對應的字串值。

格式：

CHR $ (*exprnm*)

傳回以 ***exprnm*** 的值做爲 **ASCII** 碼的對應的文字。在附錄 I 中有全部 **ASCII** 碼與各個文字的對照表；使用這個函數，您就可以產生很多爲了要控制週邊機具而又無法從鍵盤輸入的文字了。 ***exprnm*** 的值得要在 0 與 255 之間，不然就會導致 ? **ILLEGAL QUANTITY ERROR** 訊息的產生。

整數**BASIC**無此功能。

COS

傳回某個角的餘弦函數值。

格式：

COS (***exprnm***)

計算***exprnm***弧度的餘弦值。
整數**BASIC**無此功能。

EXP

算出*e*的某次冪。

格式：

EXP (***exprnm***)

計算自然對數底*e*（即2.71828183……）的***exprnm***次方。
不可以在整數**BASIC**中使用。

FN

使用一個已經由使用人定義好了的函數。

格式：

FN ***varnm*** (***exprnm***)

此地 ***varnm*** 爲函數的名稱；***exprnm*** 的值首先代入該函數定義中的每一個虛設變數內，再計算函數的值；見本章中有關 **DEF FN** 的討論。

函數不可以使用自己；換言之，***exprnm*** 中不可以再使用到 **FN *varnm***，也不可以使用到某個會使用到 **FN *varnm*** 的函數。

如果您在 **DEF FN *varnm*** 之前用到了 **FN *varnm***，您將會得到 **?UNDEF'D FUNCTION ERROR** 的訊息。

整數 **BASIC** 中不可以使用。

FRE

對一個 **APPLESOFT** 程式來說，送回目前記憶體中尚可使用的剩餘記憶體，單位是數元組（byte）。

格式：

FRE （***exprnm***）

您可以使用的記憶體位於字串區域之下，矩陣區域之上。如果還有多於 32767 個 byte 可用，**FRE** 將會傳回一個負數，您只要加上 65536 就是正確的值了。

FRE 還會把已經不用了的字串區域清除乾淨。當一個字串的值在程式執行的過程中變更了，那麼舊有的值仍然留在記憶體中，而新的值就放入字串區域，這樣就會造成浪費記憶體的現像了。爲了克服這個困難，如果您程式中大量地使用了字串，那麼建議您在程式中加上 **A＝FRE**（0）這道敍述，常常執行它。

FRE 並不會使用 ***exprnm*** 的值，不過您寫得不對就會產生錯誤。

整數 **BASIC** 中不可使用。

INT

傳回某個數的整數部份。

格式：

INT (***exprnm***)

它傳回比***exprnm***小的最大整數。
整數BASIC中不可使用。

LEFT$

傳回某個字串最左邊的一些文字。

格式：

LEFT$** (***expr$*, ***exprnm***)

傳回字串 ***expr$*** 最左邊的***exprnm***個文字。 ***exprnm*** 的值得在1到 255之間，而且 ***expr$*** 也不能超過 255 個文字。如果***exprnm***的值大過***expr$*** 的長度，那就會傳回整個字串。
整數BASIC中不可以使用。

LEN

傳回某個字串的長度。

格式：

LEN (***expr*** $)

計算 *expr* $ 中文字的個數，這包含了所有的空白以及無法印出的文字。如果 *expr* $ 超過了255個文字（有可能在 *expr* $ 是由若干字串相接而成的場合），那麼就會顯示 ?**STRING TOO LONG ERROR**訊息。

LOG

傳回某個數的自然對數值。

格式：

LOG (***exprnm***)

計算 *exprnm* 的自然對數值。如果 *exprnm* 的值小於或等於零，就會發生 ?**ILLEGAL QUANTITY ERROR**的錯誤。

整數**BASIC**中不能使用。

MID$

傳回某個字串的任何連續的一部份。

格式：

MID$ (***expr*** $, $exprnm_1$ [, $exprnm_2$])

從 *expr* $ 的第 $exprnm_1$ 個文字起，傳回 $exprnm_2$ 個文字。若 $exprnm_2$ 沒寫，則傳回從 $exprnm_1$ 起到 *expr* $ 的最後一個文字。如果 *expr* $ 的長度小於 $exprnm_1$，則傳回虛字串。如果 *expr* $ 中

從 $exprnm_1$ 起不足 $exprnm_2$ 個文字，則結果與沒寫 $exprnm_2$ 相同。*expr*$ 不能超過 255 個文字，又 $exprnm_1$ 與 $exprnm_2$ 的值也必須要在 1 與 255 之間。

整數BASIC中無此功能。

PDL

傳回指明的遊戲控制目前的值。

格式:

PDL (*exprnm*)

傳回來的值是一個 0 到 255 之間的整數，這個整數的來源有二，其一是基於在第 *exprnm* 號操縱桿上旋轉的方向，其二是來自第 *exprnm* 號遊戲控制接點上機具的電阻值。遊戲控制接點的值是在 0 與 3 之間，如果操縱桿值小於 0 或大於 255，則會顯示 ?ILLEGAL QUANTITY ERROR ，如果值在 4 與 255之間，則 PDL 會傳回一個在 0 與 255之間不可預料的結果，而且還會造成揚聲器上"嗒"的一聲或突然地變換圖型顯示方式等的副作用。

如果連續或差不多要連續地執行兩個 PDL 指令，那麼第二個值將會受到第一個的影響，所以請您注意到任意兩個 PDL 之間應該要有若干道敍述才行（比如說一個FOR-NEXT 廻圈中什麼敍述都沒有就是一個很好的例子）。

PEEK

傳回記憶體某個位置的內容。

格式：

PEEK（***memadr***）

傳回的值是以在 ***memadr*** 那兒的八個數元（ bit ）所對應的十進位值。在附錄E中有一些相當有用的記憶體位置的說明。

POS

傳回目前游標所在的行號。

格式：

POS（***exprnm***）

這個放在括號中的算式是沒有意義的，正因爲它沒有作用，所以它可以有任何合法的值。

POS 傳回的值在 0 與 39 之間；而 0 表示一列中最左邊的位置。

整數BASIC中無此函數。

RIGHT$

傳回一個字串靠右邊的文字。

格式：

RIGHT$**（***expr$*，　***exprnm***）

把 ***expr$*** 最右邊的 ***exprnm*** 個文字傳回來。***exprnm*** 的值得在 1 與 255 之間，而且 ***expr$*** 也不能多過 255 個文字；若 ***exprnm*** 的值大過 ***expr$*** 文字的個數，就會傳回整個字串。

整數BASIC中不能使用。

RND

傳回一個亂數。

格式：

RND (*exprnm*)

傳回一個亂數，這個亂數的值由 *exprnm* 的值以及您所用的 BASIC 版本而定。

在整數 BASIC 中，RND 傳回的是個整數，值在 0 與 *exprnm* 之間，可能是 0，但不會是 *exprnm*；因此 RND (1) 就一定傳回 0 了，而 RND (- 2) 的可能性就有五成爲 0，5 成爲 - 1；如果使用 RND (0)，就會導致 ***>32767 ERR 的錯誤。

在 APPLESOFT 中，RND 永遠傳回大於等於 0，而小於 1 的實數；依著 *exprnm* 的值的差異，傳回來的值就有三個類型：

若 *exprnm* 的值爲正，則每次使用 RND 時都會傳回一個不同的值；除非您起動了一串會重複的亂數序列。

當您用一個負的 *exprnm* 給 RND 時，您就起動了重復亂數列的功能；任意負數都可以使用。每一次您用不同的負值 *exprnm*，就會起動一串相同的亂數列，以後再用正數時，所得的亂數完全相同。這項功能在您測試或爲一個使用到 RND 的程式除錯時特別有用。

若 *exprnm* 的值爲 0（在 APPLESOFT 時），RND 傳回最近所產生的那個亂數（不受 CLEAR 或 NEW 的影響）。

SCRN

傳回低解析度圖型時，指定游標那兒的點的顏色。

格式：

SCRN (*col*, *row*)

如果目前的螢光幕正顯示的是文字，或者是有文字幕並且所指出的位置正好在文字幕內，則 SCRN 傳回這個文字某一個半部的顏色碼；如果 *row* 爲偶數，則傳回上半部，若 *row* 爲奇數就傳回下半部；您可以用SCRN (*a* , 2 * *b*) + 16 * SCRN (*a* , 2 * *b* + 1) 來算出在 (*a* , *b*)那個文字的ASCII 字碼，此地 *a* 要在 0 ～ 39 ， *b* 在 0～23 的範圍內；於是再用CHR$ (*n*) 就能夠求出這個文字了，此地 *n* 就是上式的值。

若 *col* 在 0 ～ 39 之間，SCRN傳回的是在 (*col*, *row*)那兒的點的顏色碼；若 *col* 在 40 ～ 47 之間，而 *row*在 0 ～ 31 之間，SCRN 就傳回在 (*col* − 40, *row*+ 16) 那個點的顏色碼；如果 *col* 在 40 ～ 47 之間，*row*在 32 ～ 47 之間，SCRN 所傳回的就是一個與螢幕沒有什麼關係的數。

當在高解析度圖型時，SCRN 所傳回來的數是依低解析度圖型區域計算，而不是用高解析度顯示區域來算的。

只有當一個括號跟在 SCRN 後面時，它才會當做保留字看。

SGN

決定一個數是正，是負，還是零。

格式：

　　SGN(*exprnm*)

SGN 傳回的值是這樣的，當 *exprnm* 之值爲正時，傳回＋1；爲負時，傳回－1；爲零時，傳回0。

SIN

傳回一個角的正弦函數值。

格式：

　　SIN(*exprnm*)

*exprnm*的單位爲弧度，這個函數計算它的正弦函數值。
整數BASIC中不能使用。

SPC

把游標在目前這一列上向右邊移若干格。

格式：

　　SPC(*exprnm*)

SPC 用來在 PRINT 敍述中印出 *exprnm* 個空格，所以游標所經過的符號都會被洗掉。

當遇到 SPC 函數時，不管游標的目前位置爲何，游標都會自此向右移動，這與 TAB 函數是不同的，後者移動的格子數目永遠自螢幕最左邊起算。

整數BASIC中不能使用。

SQR

傳回一個非負數的平方根。

格式：

SQR(***exprnm***)

如果***exprnm***是個負值，就會產生?ILLEGAL QUANTITY ERROR的訊息；請注意，SQR(***exprnm***)執行的速度要比(***exprnm***)↑(.5)來得快。

整數BASIC中不能使用。

STR$

將一個數值轉換成字串。

格式：

STR$(***exprnm***)

這個函數把 ***exprnm*** 的值轉換成字串，在這個字串中的符號與您用PRINT ***exprnm***所印出的結果相同；所以，STR$(2/3)="666666667"又STR$(2468013579)="2.46801358E+09"。如果***exprnm***的值超過了實數值的界限，就會顯示出?OVERFLOW ERROR的訊息。

整數BASIC中不能使用。

TAB

把游標向左邊移到指明了的那一行去。

格式：

TAB(*exprnm*)

在 PRINT 敍述中使用 TAB 會把游標移到目前它所在那一列的第 *exprnm* 行去，不過這要在 *exprnm* 的值所代表的位置在目前游標右方才算有效；如果是在左邊就不會有任何移動的動作了。當游標向右移時，TAB 會印出空格，所以就洗掉了它經過的位置上的內容了。

就 TAB 而言，一列上各行的數目的編號是自 1 到 255，如果 *exprnm* 的值大過輸出機具上的寬度(比如說，螢光幕上是 40 個格子)，就會移到下一列最左邊繼續向右移。若 *exprnm* 的值爲 0，TAB 就移到第 256 行，如果 *exprnm* 的值不在 0 與 255 之間就會得到 ?ILLEGAL QUANTITY ERROR 的錯誤訊息。

請參看本章中有關 HTAB(APPLESOFT)以及 TAB(整數 BASIC)的說明。

整數 BASIC 中不能使用。

TAN

傳回一個角的正切函數值。

格式：

TAN(*exprnm*)

計算一個角（單位爲弧度）的正切函數值。

整數BASIC中不能使用。

USR

將控制權交給某個機器語言副程式，再利用加法器傳回值。

格式：

USR（***exprnm***）

在位置 10 （十六進位是 0A ）那兒必須要有一個副程式，也就是說，10到 12（十六進位是 0A 到 0C ）那兒必需要有一進組合語言的 JMP 指令，跳到您的副程式起點。因爲 USR 是個函數，所以它傳回一個實數值；不管加法器中的值是什麼，一旦組合語言副程式執行 RTS 指令時就傳回加法器中的值。

APPLE Ⅱ 的監督程式中有好些個有用的機器語言副程式，請參看附錄D的說明。

也請參看本章前面對 CALL 敘述的說明，這道敘述是在整數BASIC中也可以使用的。

整數BASIC不可以使用。

VAL

VAL將一個字串轉換成數值。

格式：

VAL（***expr$***）

傳回對應著 ***expr $*** 的數值。如果 ***expr $*** 的第一個文字不是個構成數字的符號，就傳回 0 ；要不然，就一個接一個文字地取出來，一直到某個無法接受的文字爲止。能夠接受的文字包含了： 0 到 9 的數字函數、空白、一個小數點，前面可以加上一個加號或減號，另外在科學記法中還可以多出一個E字，一個可有可無的加或減號。

如果 ***expr $*** 是一個若干字串相連接的運算式；如果結果的長度多於 255 個字母，就會導致？**STRING TOO LONG ERROR** 的錯誤訊息。如果從 ***expr $*** 轉換出來的實數值超過了實數的極限，就會顯示出？**OVERFLOW ERROR**的訊息。

整數**BASIC** 中不可以使用。

附錄A

另一些函數

下面所列出來的是一些可以使用基本函數就能夠很容易組合出來的函數，當然，這並非全部，但却包含了大多數常用的公式了。就某些函數來說，有些 x 值是不能用的（比如說，令 cos(x)＝0 的點就不能計算 sec(x)），所以您必須要檢查一下所用的 x 值。

這些函數在整數BASIC中都不能使用。

ARCCOS(x) = -ATN(x/SQR(-x*x+1))+1.5707633

計算 x 的反餘弦值，此地 ABS(x)＜1。

ARCCOT(x) = -ATN(x)+1.5707633

計算 x 的反正切值。

ARCCOSH(x) = LOG(x+SQR(x*x-1))

計算雙曲式函數中，x 的反餘弦值（ABS(x)＞＝1）

ARCCOTH(x) = LOG((x+1)/(x-1))/2

計算雙曲式函數中，x 的反餘切值（ABS(x)＞1）。

ARCCSC(x) = ATN(1/SQR(x*x-1))+(SGN(x)-1)*1.5707633

計算 x 的反餘割函數值（ABS(x)＞1）。

ARCCSCH(x) = LOG((SGN(x)*SQR(x*x+1)+1)/x)

計算雙曲式函數中，反餘割函數值（x＞0）。

ARCSEC(x) = ATN(SQR(x * x − 1))+(SGN(x)− 1)*1.5707633

計算反正割函數值（ ABS(x)＞＝ 1 ）。

ARCSECH(x) = LOG((SQR(− x * x+1)+1)/x)

計算雙曲式函數中，反正割函數值（ 0＜ x ＜＝ 1 ）。

ARCSIN(x) = ATN(x/SQR(−x * x+1))

計算反正弦函數值（ ABS(x)＜ 1 ）。

ARCSINH(x) = LOG(x+SQR(x * x+1))

計算雙曲式正弦函數值。

ARCTANH(x) = LOG((1+x)/(1−x))/2

計算雙曲式函數中，反正切函數值（ ABS(x)＜ 1 ）。

COSH(x) = (EXP(x)+EXP(−x))/2

計算雙曲式餘弦函數值。

COT(x) = 1/TAN(x)

計算餘切函數值（ x＜＞0 ）。

COTH(x) = EXP(−x)/(EXP(x)−EXP(−x))*2+1

計算雙曲式餘切函數值（ x＜＞0 ）。

CSC(x) = 1/SIN(x)

計算餘割函數值（ x＜＞0 ）

CSCH(x) = 2/(EXP(x)−EXP(−x))

計算雙曲式餘割函數值（ x＜＞0 ）。

LOG_a(x) = LOG(x)/LOG(a)

計算以 a 爲底的對數值（ a＞0 ， x＞0 ）。

LOG_{10}(x) = LOG(x)/2.30258509

計算常用對數（ 10 爲底 ）值（ x＞0 ）。

MODa(x) = INT((x/a−INT((x/a))*a+.05)*SGN(x/a)

計算 x 被 a 除之後的餘數（ a ＜＞0 ）。

SEC(x) = 1/COS(x)

計算正割函數值（ x＜＞π/2 ）。

SECH(x) = 2/(EXP(x)+EXP(−x))

計算雙曲式正割函數值。

SINH(x) = (EXP(x)−EXP(−x))/2

計算雙曲式正弦函數值。

TANH(x) = −EXP(−x)/EXP(x)+EXP(−x))*2+1

計算雙曲式正切函數值。

附錄 B

編修用指令

這個附錄把Apple Ⅱ 的編修指令做一個綜合說明。

→　把游標在它現在所在的列上頭向右邊移一格，任何它經過的字都會被存入記憶體，就當是您從鍵盤鍵入的一樣，使用這個鍵不會改變螢幕上的內容。

←　把游標在它現在所在的列上頭向左邊移一格，任何它經過的字都會在記憶體中被洗掉，不過在螢幕上却還是留著不變的。

REPT　當這個鍵與另一個鍵同時按著時，就會造成該鍵的符號或動作重覆地在螢幕上出現；請注意，REPT 鍵應該後按，而需要重覆的鍵先按。

CTRL-X 表示刪掉在螢幕上現在的這一列，然後把游標移到下一列的最左邊。

ESC鍵的應用

下面有七個利用兩個鍵的組合而構成的編修指令，按鍵的方法是：先按一下Esc鍵，放開，然後再按另一個鍵。

Esc-A　把游標向右移一格，既不改變螢幕也不改變記憶體的內容。

Esc-B　把游標向左移一格，既不改變螢幕也不改變記憶體的內容。

Esc-C　把游標向下移一列，既不改變螢幕也不改變記憶體的內容。

Esc-D　把游標向上移一列，既不改變螢幕也不改變記憶體的內容。

Esc-E 把從目前游標所在的位置起到螢幕上該列的終了之間的所有文字刪掉。

Esc-F 把從目前游標的位置起到現在螢幕上最後一個文字之間的所有文字刪掉。

Esc-@ 把螢幕清乾淨，再將游標移到螢幕的左上角。

下面的四個編修指令需要有自動起動監督程式（ Autostart Monitor ）；我們可以使用按一次Esc鍵就進入了編修式（ editing mode ），然後按下不是 **I**，**J**，**K**，**M**，**REPT** 與 **SHIFT** 六個其中之一的任意鍵，就會離開編修式了。下面就是上述六個鍵的說明：

I 把游標向上移一列，但不離開編修式；

J 把游標向左移一格，但不離開編修式；

K 把游標向右移一格，但不離開編修式；

M 把游標向下移一列，但不離開編修式。

附錄C

錯誤訊息

錯誤訊息可以分成三部份：整數BASIC，APPLESOFT 與 DOS 三項，每一項我們都按字母順序列出來。

DOS與大多數APPLESOFT錯誤訊息都有一個對應的錯誤代碼，因此一旦有錯誤發生，而造成ONERR GOTO 敍述有所動作之後，您可以在 222 那個位置找到所要的錯誤碼。本附錄末的表 C-1 列出了所有錯誤代碼。

整數BASIC錯誤訊息

***** > 255 ERR**

一個應該在 0 與 255 之間的值超出了範圍。

***** > 32767 ERR**

已經算出或輸入了某個大於 32767 或小於 -32767 的數值。

***** 16 FORS ERR**

已經有十六個的FOR同時在進行。

***** 16 GOSUB ERR**

已經有十七個GOSUB在執行了，但從第一個到現在還沒有執

行任何RETURN 敍述。

*** BAD BRANCH ERR

嘗試轉向到一道不存在的列編號。

*** BAD NEXT ERR

執行到一個不與任何 FOR 對應的NEXT 敍述。

*** BAD RETURN ERR

執行到一道不與任何GOSUB 對應的 RETURN 敍述。

*** DIM ERR

同樣名稱的陣列被重定了兩次維數了。

*** MEM FULL ERR

記憶體已經不夠用了。

*** NO END ERR

程式中最後一道被執行的敍述不是END 。

*** RANGE ERR

這有兩種情況：一是您參考到某個陣列的元素時，足碼值小於零或大於陣列的容量，二是HLIN，VLIN，PLOT，TAB 或VTAB 敍述中的值超出了既有的範圍。

RETYPE LINE

由 INPUT 所產生的錯誤，這會先產生一道錯誤，然後才是這道訊息，請您把那一列重新打過。

*** STR ERR

您嘗試執行一個不合法的字串運算。

*** STR OVFL ERR

一個字串中已經被您存入比它容量還多的字符了。

*** STNTAX ERR

這指的是拼字，標點，有關書寫次序，以及不能由其它訊息所能涵蓋的錯誤。

*** TOO LONG ERR

一列上頭已經超過了128個文字，或者是已經超過了 12 層括號了。

APPLESOFT 錯誤訊息

?BAD SUBSCRIPT ERROR

在參考陣列的元素時，所使用的足碼數目不正確，或者是足碼的值超過了既定的容量。錯誤代碼是 107 。

?CAN'T CONTINUE ERROR

在根本沒有程式，或者是已經有個嚴重錯誤出現，或者已經修改過程式之後，嚐試繼續程式的執行（使用CONT指令）。

?DIVISION BY ZERO ERROR

嚐試用值爲零的算式做除數。錯誤代碼是 133 。

?FORMULA TOO COMPLEX ERROR

如果像IF *string* THEN這樣的敍述已經執行了兩次以上的話，就會產生這一道訊息。錯誤代碼爲 191 。

?ILLEGAL DIRECT ERROR

在直接式中執行 INPUT, DEF FN 或 GET 指令。

?ILLEGAL QUANTITY ERROR

在一個字串函數，數值函數，描圖敍述等的敍述中，所用到的數值超過了能夠接受的範圍。錯誤 53 。

?NEXT WITHOUT FOR ERROR

執行到一道沒有FOR對應的NEXT敍述。請注意，如果目前沒有 FOR 已經執行，然後却執行了一道後面沒有帶變數的 NEXT，就必定會產生這一道訊息。錯誤代碼爲 0 。

?OUT OF DATA ERROR

在輸入時，DATA敍述中已經沒有可用的變數了。錯誤代碼 42 。

?OUT OF MEMORY ERROR

下面都是可能的成因：程式太大，變數太多，FOR 廻圈的層次超過 10, 副程式層次超過 24, 括號的深度超過 36 層，LOMEM：定得太高，HIMEN ：定得太低等等。錯誤代碼 77。

?OVERFLOW ERROR

您鍵入或程式計算出來的數太大，或太小了；機器能夠接受的範圍是 − 1.7E+ 38與1.7E+ 38 之間。錯誤代碼 69 。

?REDIM'D ARRAY ERROR

嚐試用一個DIM 敍述來改變一個已經定好維數陣列的維數；最常見的錯誤在於某個陣列已經由約定給出了維數，但還嚐試去更改它。錯誤代碼爲 12 。

?RETURN WITHOUT GOSUB ERROR

執行到一個沒有GOSUB與之對應的RETURN 敍述。錯誤代碼 22 。

?STRING TOO LONG ERROR

在把各個字串相連接的時候，所得出來字串的長度超過 255個字母。錯誤代碼176 。

?SYNTAX ERROR

在拼字的時候，或者是標點的使用，次序等等不對，以及任何不能由其它訊息所能涵蓋的錯誤都會產生這一項訊息。錯誤代碼 16 。

?TYPE MISMATCH ERROR

當某個所在需要字串時，但却使用數値運算式或變數；或者是需要數値的式子或變數時，却用字串，就會產生這一道訊息。另外，在設定敍述兩邊的資料型别不符時也會有此訊息發生。

錯誤代碼 163 。

?UNDEF'D FUNCTION FRROR

您使用到一個您並沒有定義出來的函數。錯誤代碼 224 。

?UNDEF'D STATEMENT ERROR

程式中嚐試轉向到一個不存在的列編號。錯誤代碼 90 。

DOS錯誤訊息

DISK FULL

磁碟片已經裝滿了，但您還嚐試把資料儲存進去。錯誤代號 9。

END OF DATA

一個文字檔（ text file ）已經沒有資料了，但您還嚐試要去讀取資料。錯誤代碼 5 。

FILE LOCKED

您嚐試使用 **SAVE**, **BSAVE**, **WRITE**, **DELETE** 或 **RENAME** 在一個已經鎖住了的檔案上頭作業。錯誤代碼 10 。

FILE NOT FOUND

您參考到一個不在磁碟上頭的檔案。這個錯誤只有當某個 **DOS** 指令參考到某檔案，但又不產生該檔案時才會發生。錯誤代碼 6 。

FILE TYPE MISMATCH

您用 **DOS** 指令參考到一個型態不正確的檔案。好比說，**LOAD**, **RUN** 與 **SAVE** 指令只能用在程式檔，**CHAIN** 只能用在整數 **BASIC** 的程式檔；而 **OPEN**, **READ**, **WRITE**, **APPEND**, **POSITION** 與 **EXEC** 指令只能用在文字檔。最後，**BLOAD**, **BSAVE** 與 **BRUN** 只能用在二進位檔。錯誤代碼 13 。

I/O ERROR

在磁碟上取出或儲存某個檔案不成功。常見的成因如次：磁碟機上的門開著，磁碟片尚未啓用，磁碟機上沒有磁碟片，或磁碟片可能有所損壞等等。錯誤代碼 8 。

LANGUAGE NOT AVAILABLE

當您用 **FP** 或 **INT** 改變目前系統中的語言，而這套語言却又不在 **ROM** 或者是磁碟中；或者是要抄入或執行某個使用目前系統中沒有這種語言的程式時，就會發生這種錯誤。錯誤代碼爲 1 。

NO BUFFERS AVAILABLE

當您需要一個檔案緩衝區時已經沒有可用的了；錯誤代碼 12。

表 C-1 錯誤代碼

PEEK(222)	說明	語言
0	沒有 **FOR** 對應的 **NEXT**	Applesoft
1	沒有這套語言	DOS
2 or 3	範圍錯誤	DOS
4	被保護的檔案	DOS
5	資料終了	DOS
6	找不到檔案	DOS
7	磁碟片不對	DOS
8	I/O 錯誤	DOS
9	磁碟滿了	DOS
10	檔案已被鎖住	DOS
11	語法錯誤	DOS
12	沒有緩衝區可用	DOS
13	檔案型態不相配	DOS
14	程式太大	DOS

（續下頁）

（接上頁）

PEEK(222)	說　　明	語　　言
15	不是直接式指令	DOS
16	語法錯誤	Applesoft
22	沒有GOSUB對應的RETURN	Applesoft
42	沒有DATA敘述了	Applesoft
53	不正確的量	Applesoft
69	溢載	Applesoft
77	記憶體不夠用	Applesoft
90	沒有定義的敘述	Applesoft
107	不正確的足碼	Applesoft
120	更改行列的度量	Applesoft
133	用零除	Applesoft
163	型別不相配	Applesoft
176	字串太長	Applesoft
191	運算式太複雜	Applesoft
224	沒有定義的正數	Applesoft
254	對INPUT不正確的回應	Applesoft
255	已經按了CTRL-C	Applesoft

NOT DIRECT COMMAND

下面的幾道DOS 指令在間接式時是只能用在PRINT敘述中的：APPEND,OPEN,POSITION,READ,WRITE，如果不這樣用就會發生錯誤。錯誤代碼15。

PROGRAM TOO LARGE

當一個DOS 指令把一個磁碟檔抄入APPLE Ⅱ 記憶體，却發現記憶體不足以容納它時，就會產生這個錯誤。錯誤代碼14。

RANGE ERROR

在一道DOS 指令中某個參數的值超過了指定的範圍；好比說，D（磁碟機）後的參數必須要是 1 或 2 就是一例。錯誤代碼 2 或 3 。

SYNTAX ERROR

某個DOS 指令中有拼字拼錯，標點用錯，或次序不對等等。錯誤代碼 11 。

VOLUME MISMATCH

DOS 指令中V（磁碟片）參數的值與磁碟片上所記錄者不符。錯誤代碼 7 。

WRITE PROTECTED

您嘗試在一個已經被保護起來了的磁碟上使用 SAVE ,BSAVE 或WRITE 指令。錯誤代碼 4 。

附錄 D

固有的一些副程式

下面的兩個表將一些有用的機器語言副程式列出來；表D－1依照它們的功能來列，但並沒有它們詳細的資料。您可以在表D－1內發現這些副程式的入口位置，而在表D－2的第一行中對應著它們詳細的操作情形。

表D－2依照入口的順序排列這些副程式，表中的第三行列出這個副程式需要使用到的暫存器的起始值；第四行則列出在這個副程式被執行後會影響到那些暫存器。

大多數的這些副程式都在**BASIC**中有相同功效的命令，或者可以在**BASIC**程式中用**CALL** 命令使用它們，這種相通的情形在表D－2最後一行中也詳列出來。有些被標記爲**A**字的**BASIC**命令是只能被使用在**APPLESOFT**中的。

然而有一些副程式却沒有相似功能的**BASIC**命令，也不能在**BASIC**程式中使用**CALL** 指令使用它們，這是因爲它們所使用的暫存器必須在執行之前先被放入某些特定的值。至於處理這個問題的方法，依照**BASIC**型式的不同而有不同的方式。

整數**BASIC**提供一種簡單的解決方法，首先執行**CALL－182**命令將暫存器現在的值存入**RAM**內，然後使用**POKE**指令將暫存器**A**所需要的值放入記憶體69的位置去，**X**暫存器所需要的值放入70的位置內

，Y暫存器所需要的值放入71的位置內。然後執行一個CALL－193的命令將這些值存回暫存器，然後再使用一個CALL用命令執行這個副程式。

這個方法在APPLESOFT 中並無效用，您必須設計一個機器語言的副程式將這些暫存器需要的值放入暫存器內；然後執行一個 JSR 的指令跳到您所要使用的副程式的起始位置而執行它。

表 D-1 由功能分類固有的副程式

	功　能	進入點
低解析度圖形	繪出一個低解析度圖形的點。	$F800
	繪出一條低解析度圖形的水平綫。	$F819
	繪出一條低解析度圖形的垂直綫。	$F828
	將低解析度圖形的48列清除為黑色。（如果在文字資料情況時，則變成黑白相反的“@”符號）	$F832
	將低解度圖形的上面40列清爲黑色（或是黑白相反的“@”）	$F836
	將現有的低解析度圖形的顏色增加三種。	$F85F
	設定低解析度圖形的顏色。	$F864
	將低解析度圖形中某一點的顏色讀出。	$F871
	設定低解析度圖形的情況，並將螢幕清除，同時設定四行位置的文字資料幕。	$F840
輸　入	當游標閃動時等待鍵盤的輸入，並且使用$4E及$4F 位置內的值做爲亂數產生器的原始數字。	$FD1B
	與上面功能相同，只是也可以接受編輯碼。	$FD35

	功　　能	進入點
	將一個游標回頭訊號送回到螢光幕，然後允許在同一行內輸入至多 256 個字。	$FD67
輸　出	送出三個空白到被選定的輸出設備內。	$F948
	送出由 1 到 256 個的空白到指定的輸出設備內。	$F94A
	送出一個游標回頭的訊號並且跳到下一個顯示行的位置。	$FC62
	輸出一個字到被選定的輸出設備內。	$FDED
	輸出一個字到文字資料幕上。	$FDF0
BELL 輸　出	送出BELL字（ASCII 碼 7 ）到被選定的輸出設備內。	$FBD9
	使發聲器發出 1 / 10 秒"嗶"的聲音。	$FBE4
	印出ERR 的訊息並且使發聲器發出"嗶"的聲音。	$FF2D
	使發聲器發出"嗶"的聲音。	$FF3A
文字資料幕	將APPLE II 的螢光幕設定為24列、40行。	$FB2F
	將文字資料幕的位置往上捲一行。	$FC70
游標的控制	送一個回頭的訊號到螢光幕上，修正游標的位置。	$FC10
	將游標往上移動一行，如果游標已經在螢幕的最上端了，則不做任何移動。	$FC1A
	將游標往下移動一行，但並不改變它的水平位置。如果游標在螢幕的最底端，則將文字資料幕往上捲一行。	$FC66

	功　能	進入點
螢幕的清除	將文字資料幕自游標所在位置開始到螢幕右下角爲止的區域清除乾淨。	$FC42
	將文字資料幕自暫存器所指定的座標開始到螢幕右下角爲止的區域清除乾淨。	$FC46
	將顯示文字資料的螢光幕清除掉，並將游標移到螢幕的左上角。	$FC58
	將自游標位置開始這一行的文字資料全部清除。	$FC9C
顯示的情形	將螢光幕顯示設定爲黑字白底。	$FE80
	將螢光幕顯示設定爲正常情況（白字黑底）。	$FE84
印出暫存器之內容	將X及Y暫存器的內容輸出到被選擇的輸出設備上（按照YYXX的格式）。	$F940
	將A及Y暫存器的內容輸出到被選擇的輸出設備上（按照AAXX的格式）。	$F941
	將X暫存器的內容輸出到被選擇的輸出設備上。	$F944
	將A暫存器的內容輸出到被選擇的輸出設備上。	$FDDA
移動暫存器的內容	重新將暫存器的值存回暫存器內（只在由$FF4A開始的副程式先被執行後才有效）。	$FF3F
	將暫存器的內容存在第零頁的位置內。	$FF4A
其　他	讀取一個遊戲控制器的情況。	$FB1E
	執行一個延遲執行的廻圈。	$FCA8

	功　　能	進入點
	回到BASIC 語言，並消除記憶體內的變數及程式。	$FEB0
	監督系統的進入點。	$FF69

表 D-2. 由輸入點排列固有的副型式

輸入點	功　　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敘述
$F800	在低解析度圖形第一頁內劃出一個點。	將列數放在暫存器A內，行數放在Y內。	沒　有	PLOT
$F819	繪出一條低解析度的水平綫。	將列數放在暫存器A內，左邊行數放在Y內，右邊行數放在記憶體位置45內。	A，Y	HLIN
$F828	繪出一條低解析度的垂直綫。	行數在Y內，上面列數在A內，下邊列數在記憶體位置 45 內。	沒　有	VLIN
$F832	將48列的低解析度圖形顯示設定爲黑色（如果在文字資	沒有	A，Y	CALL-1998

輸入點	功　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敍述
	料形式時，則設定爲"@")			
$F836	將低解析度圖形清除，但不變動文字資料幕。	沒　有	A，Y	GR（參看$FB40）
$F85F	將現有的低解析度圖形的顏色增加三種。	沒　有	A	CALL-1953
$F864	設定低解析度圖形的顏色。	代表顏色的數字放在暫存器A內。	A	COLOR
$F871	將低解析度圖形中某一點的顏色讀出。	列數在暫存器A內，行數在Y內。	A（存著代表顏色的數字）	SCRN
$F940	將X及Y暫存器的內容輸出到被選擇的輸出設備上（按照YYXX的格式）	沒　有	沒　有	CALL-1728
$F941	將A及Y暫存器的內容輸出到被選擇的輸出設備上（按照AAXX的格式)。	沒　有	沒　有	CALL-1727
$F944	將X暫存器的內容輸出到被選擇的輸出設備上。	沒　有	沒　有	CALL-1724

輸入點	功　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敍述
$F948	送出三個空白到被選定的輸出設備內（由CSW的資料而決定）。	沒　有	X，A	CALL-1720
$F94A	送出由1到250個的空白到指定的輸出設備內。	空白的數目存在暫存器A內（存入0表示輸出256個空白）。	沒　有	SPC( )[A] CALL-1718
$FB1E	讀取遊戲控制器0，1，2及3的情況。	遊戲控制器的號碼在暫存器X內。	Y內存有0－FF的資料。A內資料被清除。	PDL( )
$FB2F	將APPLE II的螢光幕設定為24列、40行。	沒　有	A	TEXT
$FB40	設定低解析度圖形的情況，並將螢幕清除，同時設定四行位置的文字資料幕。	沒　有	A，Y	GR
$FBD9	送出BELL字（ASCII碼7）到	沒　有	A，Y	CALL-1063

輸入點	功　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敍述
	被選定的輸出設備內。			
$FBE4	使發聲器發出 1/10 秒"嗶"的聲音。	沒　有	A，Y	CALL-1052
$FC10	送一個回頭的訊號到螢幕上，修正游標的位置。	沒　有	A	CALL-1008
$FC1A	將游標往上移動一行，如果游標已經在螢幕的最上端，則不做任何移動。	沒　有	A	CALL-998
$FC42	將文字資料幕自游標所在位置開始到螢幕右下角爲止的區域清除乾淨。	沒　有	A，Y	CALL-958
$FC46	將文字資料幕自暫存器所指定的座標開始到螢幕右下角爲止的區域清除乾淨。	行數在暫存器Y內，列數在A內。	A，Y	CALL-954
$FC58	將顯示文字資料的螢光幕清除掉，並將游標移到螢幕的左上角。	沒　有	A，Y	HOME [A] CALL-936
$1C62	送出一個游標回頭	沒　有	A，Y	CALL-

輸入點	功　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敘述
	的訊號，並且跳到下一個顯示行的位置。			926
$FC66	將游標往下移動一行，但並不改變它的水平位置。如果游標在螢幕的最底端，則將文字資料幕往上捲一行。	沒　有	A , Y	CALL-922
$FC70	將文字資料幕往上捲一行。	沒　有	A , Y	CALL-912
$FC9C	將自游標位置開始這一行的文字資料全部清除。	沒　有	A , Y	CALL-868
$FCAB	執行一個延遲的廻圈，它的長度是 $0.5(5X^2+27X+26)$ microseconds.	X值在暫存器A內	A	CALL-856
$FD0C	當游標閃動時等待鍵盤的輸入，並且使用 78 及 79 內的值做爲亂數產生器的原始數字。	沒　有	字被輸入A內。X , Y	CALL-756

輸入點	功　　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敍述
$FD35	與上面的功能相同，只是也可以接受編輯碼。	沒　有	字被輸入A內。X，Y	CALL-715
$FD67	將一個游標回頭送回到螢光幕，然後允許同一行內輸入256個字。	顯示字在記憶體位置51內。	Y，A X內存有輸入資料的長度。資料輸入由$200的位置開始	INPUT
$FDDA	將A暫存器內的值以兩位十六進位數字的表示方法輸出。	資料在A內。	A	CALL-550
$FDED	輸出一個字到被選定的輸出設備內。	字在A內。	沒　有	PRINT
$FDF0	輸出一個字到文字資料幕上。	字在A內。	沒　有	PRINT
$FE80	將螢光幕顯示設定爲黑字白底。	沒　有	Y	INVERSE[A] CALL-384
$FE84	將螢光幕顯示設定爲正常情況（白字黑底）。	沒　有	Y	NORMAL[A] CALL-380
$FEB0	回到BASIC語言，並消除記憶體內的變數及程式。	沒　有	A，X，Y	CALL-336
$FF2D	印出ERR 的訊息並且使發聲器發出“嗶”的聲音。	沒　有	A	CALL-211

輸入點	功　能	使用前必須使用的暫存器	受到影響的暫存器	相同功能的BASIC敍述
$FF3A	使發聲器發出"嗶"的聲音。	沒　有	A	CALL-198
$FF3F	重新將暫存器的值存回暫存器內（只在由 **$FF4A** 開始的副程式先被執行後才有效）。	沒　有	暫存器內容由以下的位置內存回：**A**：69（$45）**S**：72（$48）**X**：70（$46）**Y**：71（$47）先入後出暫存器：73（$49）	CALL-193
$FF4A	將暫存器的內容存在第零頁的位置內：**A**：69（**$**45）**S**：72（**$**48）**X**：70（**$**46）先入後出暫存器：73（**$**49）**Y**：71（**$**47）	沒　有	沒　有	CALL-182
$FF69	監督系統的進入點。	沒　有	沒　有	CALL-151

[A] 表示只能在 **APPLESOFT** 語言內使用的 **BASIC** 命令。

附錄 E

有用的PEEK及POKE位置

下面所列的記憶體位置都是使用十進位的數字表示的，它們的值都較32767爲小，至於較32767爲大的記憶體位置都是使用負值來表示的。當然，它們也可以使用正數表示同樣的位置，您只需將下面的負值加上65536就是它相同位置的正數表示了（例如，65536 － 16384 ＝ 49152 ）。

下面所描述的一些功能只要您去使用它就會產生預期的功效，也就是說，任何時間您只要使用 PEEK 敍述去處理這個位置，那麼就能夠照您所期望的替您做事了。一個 POKE 敍述也會使這個動作發生，但由於 APPLE Ⅱ 微處理機的特性，一個 POKE 敍述實際上使同樣的動作發生兩次，在這種情形時，一個 POKE 敍述就等於兩個 PEEK 敍述。一般說來，這並沒有多大的差別，但在如－16336位置（使發聲器發聲）時，就有很大的區別了。

文字資料幕及游標的控制

32 文字資料幕的左邊極限

設定文字資料幕的左邊行數， PEEK 送回由 0 到 39 之間的數字，0 表示螢幕的最左端。改變這個位置內的值對於文字資

在文字資料幕的右邊極限，但游標只在印出一個字之後就離開這個位置。絕不要將一個與位置 32 內的值相加超過 39 的值放入這個位置內。

這個 PEEK 指令與 APPLESOFT 中的 POS 指令相同。

37　游標的垂直位置

設定游標的垂直位置，PEEK 送回一個在 0 與 23 之間的值，這個值與螢光幕的上端相關（並不是文字資料幕的上端）。絕不要將一個超過 23 的值放入這個位置內。

處理錯誤的位置

216　錯誤指標

指出 ONERR GOTO 是否發生作用，如果第 7 個數元是 1，那麼只要 ONERR GOTO 敍述被執行到時，執行就會轉移到這個敍述所指定的行號去，POKE 一個較 128 爲小的值到這個位置去時，就能使 ONERR GOTO 敍述不被執行。

218及219　引起錯誤的行號

當一個錯誤發生而使得 ONERR GOTO 中所指出的行號被執行時，這兩個位置內就儲存著發生錯誤敍述的行號。而這個行號可以由 PEEK（219）*256+PEEK（218）得到。

222　錯誤的型式碼

指出發生那一種型式的錯誤；錯誤碼及它們的說明在附錄C中有詳細的說明。

鍵盤位置

−16384　由鍵盤輸入資料

由鍵盤讀入資料，如果在這個位置內的值較 127（第 7 個數元

為1）為大，則表示剛剛有一個鍵被按下了。您可以將這個值減去 128 而得到輸入字的 **ASCII** 碼。

−16368　鍵盤指標

將位置 −16384 的第 7 個數元設定為 0 ，使得下一個字可以被輸入。

"CLICK"輸出位置

−16352　磁帶喀嗒聲

由磁帶輸出頭處產生一個可聽到的喀嗒聲。

−16336　發聲器喀嗒聲

由發聲器產生一個喀嗒聲。

顯示開關

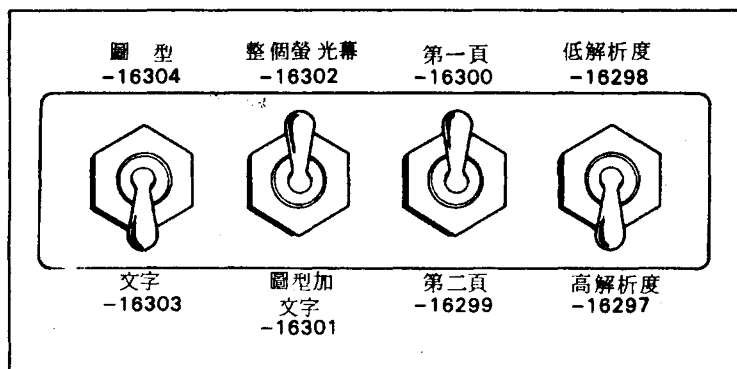
這一節中所列的記憶體位置設定一些開關使得能夠決定顯示情形的特性，但它們並不是眞正的開關，只能使用 **PEEK** 及 **POKE** 命令來設定它們。它們分別有四種不同的開關，可以設定在兩種不同的情形，就如圖 E − 1 所顯示的。當您選擇文字資料的顯示情形時，另外一個可以影響顯示的開關就是第一頁／第二頁的開關。

−16304　選擇圖形顯示情形

選擇圖形顯示情形，圖形螢幕並沒有被清除為黑色。圖形顯示情形可以是高解析度或低解析度、第一頁或第二頁、全螢幕的圖形或混合圖形與文字資料的顯示，這些特性可以由其他的記憶體位置來決定。

−16303　選擇文字顯示情形

選擇文字顯示情形，這些文字資料可以存在第一頁或第二頁內，這也是由其他的記憶體位置決定。



圖E－I PEEK/POKE 圖形及文字資料的位置

－16302 選擇全螢幕的顯示情形

選擇全螢幕的顯示情形；如果螢幕顯示情形是文字資料，那麼除非－16304的位置被處理過後，這個情形才能顯示出來。

－16301 選擇圖形及文字資料的顯示

在螢幕的下方建立一個佔有四行位置的文字資料幕。如果螢幕顯示情形是文字資料，那麼除非－16304的位置先被處理，否則這種情形將看不見。

－16300 選擇第一頁的顯示頁

選擇第一頁的圖形或是文字資料頁。

－16299 選擇第二頁的顯示頁

選擇第二頁的圖形或是文字資料頁。

－16298 選擇低解析度的圖形顯示

選擇低解析度的圖形顯示。如果螢光幕顯示情形是文字資料，那麼除非－16304的位置先被處理，否則這種情形將無法看到。

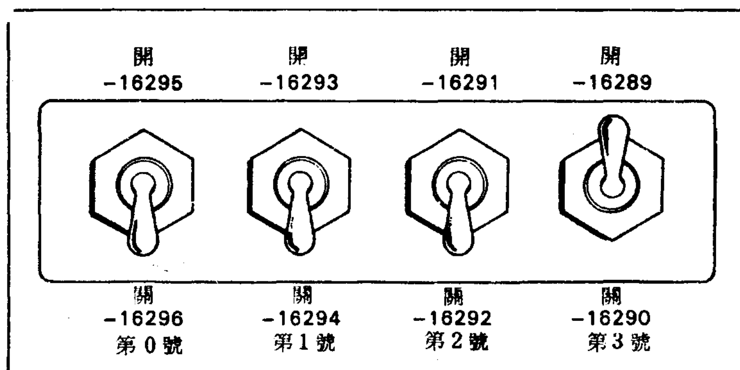
－16297 選擇高解析度圖形顯示

選擇高解析度圖形顯示。如果螢幕顯示情形是文字資料，那麼

除非－16304的位置先被處理，否則這種情形將無法看到。

控制遊戲的位置

在這一節中所列出的位置會將控制遊戲的開關打開或關閉，感應那一個按鈕被按了或沒有按，以及造成正常的輸出。圖E－2就是這些控制位置的處理情形。



圖E－2　控制遊戲的輸出情況

－16296　將第0個控制器關閉

將第0個控制器的輸出關閉，在遊戲控制器的連接位置上的第15個點的電壓被設定爲0伏（TTL *high*）。

－16295　將第0個控制器打開

將第0個控制器的輸出打開，在遊戲控制器的連接位置上的第15個點的電壓被設定爲＋5伏（TTL *low*）。

－16294　將第1個控制器關閉

將第1個控制器的輸出關閉，在遊戲控制器的連接位置的第

14 個點的電壓被設定為 0 伏（ **TTL** ***high*** ）

−16293 將第 1 個控制器打開

將第 1 個控制器的輸出打開，在遊戲控制器的連接位置上的第 14 個點的電壓被設定為 + 5 伏（ **TTL** ***low*** ）。

−16292 將第 2 個控制器關閉

將第 2 個控制器的輸出關閉，在遊戲控制器的連接位置上的第 13 個點的電壓被設定為 0 伏（ **TTL** ***high*** ）。

−16291 將第 2 個控制器打開

將第 2 個控制器的輸出打開，在遊戲控制器的連接位置上的第 13 個點的電壓被設定為 + 5 伏（ **TTL** ***low*** ）。

−16290 將第 3 個控制器關閉

將第 3 個控制器的輸出關閉，在遊戲控制器的連接位置上的第 12 個點的電壓被設定為 0 伏（ **TTL** ***high*** ）。

−16289 將第 3 個控制器打開

將第 3 個控制器的輸出打開，在遊戲控制器的連接位置上的第 12 個點的電壓被設定為 + 5 伏（ **TTL** ***low*** ）。

−16287 讀取按鈕 0 的資料

當位於遊戲控制器 0 上的按鈕被壓下時，在這個位置內的值超過 127 ；當沒有被壓下時，這個位置內的值是 127 或是小於 127 。第 0 個按鈕與遊戲控制器的連接位置的第 2 個點連接。

−16286 讀取按鈕 1 的資料

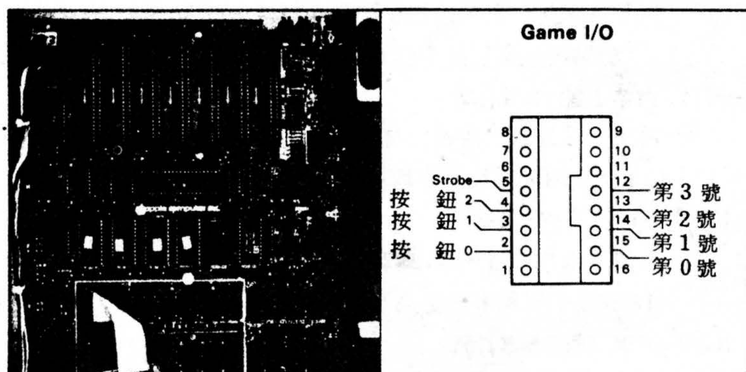
當位於遊戲控制器 1 上的按鈕被壓下時，在這個位置內的值超過 127 ；當沒有被壓下時，這個位置內的值是小於或等於 127 。第 1 個按鈕與遊戲控制器的連接位置的第 3 個點連接。

−16285 讀取按鈕 2 的資料

當位於遊戲控制器 2 上的按鈕被壓下時，在這個位置內的值超過了 127 ；當沒有被壓下時，這個位置內的值是小於或等於 127 。第 2 個按鈕與遊戲控制器的連接位置的第 4 個點連接。

−16272　STROBE的輸出

正常情形下，遊戲控制器的連接位置的第5點是＋5伏，如果您 **PEEK** 記憶體−16285的位置，它的電壓就會掉到0伏大約半個***microsecond***。**POKE** 將開動這條綫兩次。



圖E−3　控制遊戲的輸入及輸出

附錄 F

BASIC的保留字

當APPLE Ⅱ 遇到下面所列的字時，它就把它們當做是一個BASIC 的命令，敍述或是函數，唯一的例外，是當這些字被包含在引號之內時。所以在您的變數名字內決不可以使用保留字，尤其對於短的保留字特別要注意。

您可以在保留字之間加入空白，APPLE Ⅱ 會自動地將它們消除掉。

整數BASIC

ABS	END	LET	PDL	SAVE
AND	FOR	LIST	PEEK	SCRN
ASC	GOSUB	LOAD	PLOT	SGN
AT	GOTO	LOMEM:	POKE	STEP
AUTO	GR	MAN	POP	TAB
CALL	HIMEM:	MOD	PRINT	TEXT
COLOR=	HLIN	NEW	PR#	THEN
CON	IF	NEXT	REM	TO
DEL	IN#	NOT	RETURN	TRACE
DIM	INPUT	NOTRACE	RND	VLIN
DSP	LEN	OR	RUN	VTAB

APPLESOFT

APPLESOFT 中的保留字都使用代號代表它們——每一個字只佔用一個數元組的程式位置。代號與保留字同時在下面對照著列出來，在附錄 I 中您還可以依照數字的順序參照它們的對照情形。

如果有下面的情形，APPLESOFT 則無法認識TO這個保留字:

1. 在TO之前的第一個並非空白的字是A。
2. T與O之間有一個或多個的空白。

ABS	(212)	HTAB	(150)	REM	(178)
AND	(205)	IF	(173)	RESTORE	(174)
ASC	(230)	IN#	(139)	RESUME	(166)
AT	(197)	INPUT	(132)	RETURN	(177)
ATN	(225)	INT	(211)	RIGHT$	(233)
CALL	(140)	INVERSE	(158)	RND	(219)
CHR$	(231)	LEFT$	(232)	ROT=	(152)
CLEAR	(189)	LEN	(227)	RUN	(172)
COLOR=	(160)	LET	(170)	SAVE	(183)
CONT	(187)	LIST	(188)	SCALE=	(153)
COS	(222)	LOAD	(182)	SCRN(	(215)
DATA	(131)	LOG	(220)	SGN	(210)
DEF	(184)	LOMEM:	(164)	SHLOAD	(154)
DEL	(133)	MID$	(234)	SIN	(223)
DIM	(134)	NEW	(191)	SPC(	(195)
END	(128)	NEXT	(130)	SPEED=	(169)
EXP	(221)	NORMAL	(157)	SQR	(218)
FLASH	(159)	NOT	(198)	STEP	(199)
FN	(194)	NOTRACE	(156)	STOP	(179)
FOR	(129)	ON	(180)	STORE	(168)
FRE	(214)	ONERR	(165)	STR$	(228)
GET	(190)	OR	(206)	TAB(	(192)
GOSUB	(176)	PDL	(216)	TAN	(224)
GOTO	(171)	PEEK	(226)	TEXT	(137)
GR	(136)	PLOT	(141)	THEN	(196)
HCOLOR=	(146)	POKE	(185)	TO	(193)
HGR	(145)	POP	(161)	TRACE	(155)
HGR2	(144)	POS	(217)	USR	(213)
HIMEM:	(163)	PRINT	(186)	VAL	(229)

HLIN	(142)	PR#	(138)	VLIN	(143)
HOME	(151)	READ	(135)	VTAB	(162)
HPLOT	(147)	RECALL	(167)	WAIT	(181)
				XDRAW	(149)

DOS

DOS 的命令只有在直接式執行的情況下或是在 **PRINT** 敘述中前面跟著一個**C**TRL - **D**字（**ASCII** 碼的 4 ）時，才被認爲是保留字。

APPEND	CHAIN	INIT	POSITION	SAVE
BLOAD	CLOSE	LOAD	READ	UNLOCK
BRUN	DELETE	LOCK	RENAME	VERIFY
BSAVE	EXEC	OPEN	RUN	WRITE

附錄 G

記憶體位置的使用

記憶體位置的一般結構

APPLE Ⅱ的記憶體被分爲三個主要的部份：讀／寫記憶體（RAM），僅讀記憶體（ROM）及輸入／輸出位置（I／O）。記憶體位置 0 到 49151（$BFFF）的區域就是RAM，49152（$C000 到 53247（$CFFF）的區域爲ROM。您的系統並不一定需要這些位置的實際硬體。例如，如果您只有 16K 的RAM，那麼由 16384（$4000）到 49151（$BFFF）的記憶體位置就不會被使用。

表G－1就是記憶體在APPLE Ⅱ內的分佈情形，需要注意的是有兩塊沒有被佔用的記憶體區域，兩旁由兩頁高解析度頁所包圍著，系統指標 LOMEM：記錄著這塊區域的下限，而 HIMEM：則記錄著這塊區域的上限。這一塊區域的讀／寫記憶體可以供您任意地使用，它們可以被用來儲存由磁帶或磁碟輸入的APPLESOFT 編譯程式，DOS 、高解析度圖形、以及您的 BASIC 程式及變數。

表G－1 BASIC記憶體的組織

位置		記憶體的型式	用途
十進位	十六進位		
0- 255	$ 0-$0FF	RAM	系統程式
256- 511	$ 100-$1FF	RAM	系統堆疊
512- 767	$ 200-$2FF	RAM	鍵盤輸入緩衝區
768- 1023	$ 300-$3FF	RAM	監督程式使用之向量位置
1024- 2047	$ 400-$7FF	RAM	文字與低解析度圖型第一頁
2048- 3071	$ 800-$BFF	RAM	文字與低解析度圖型第二頁
3072- 8191	$C 00-$1FFF	RAM	沒有設定用途
8192-16383	$ 2000-$3FFF	RAM	高解析度圖型第一頁
16384-24575	$ 4000-$5FFF	RAM	高解析度圖型第二頁
24576-49151	$ 6000-$BFFF	RAM	沒有設定用途
49152-49279	$C000-$C07F	I/O	特殊的固定用途
49280-49407	$C080-$C0FF	I/O	週邊卡I/O位置
49408-51199	$C100-$C7FF	I/O	週邊卡記憶體
51200-53247	$C800-$CFFF	I/O	週邊卡擴充記憶體
53248-65535	$D000-$FFFF	ROM	整數BASIC、APPLESOFT、監督程式、自動啓動監督程式……等等

BASIC語言的編譯程式

如您由表G－1所看到的，整數 BASIC 的編譯程式被存在 ROM 內；如果您的系統中具有 APPLESOFT 介面卡或是語言系統卡，那麼您的 APPLESOFT 編譯程式也被儲存在 ROM 內；否則，您的APPLESOFT編譯程式大約佔據由 2048（$800）位置開始的 10K 位置。

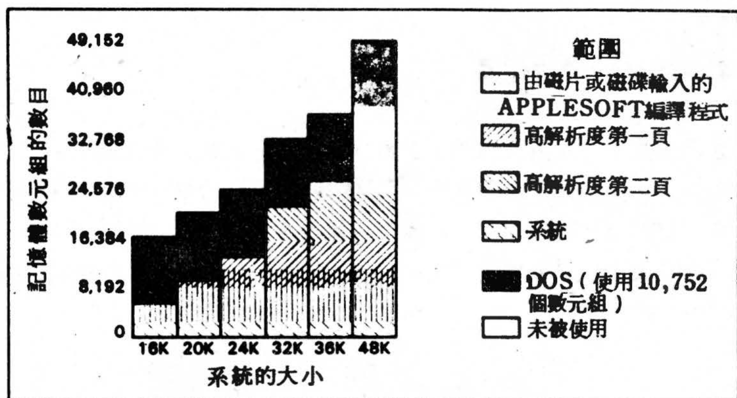
DOS的記憶體需求

如果您想要使用 DOS，那麼您至少需要有 16K 的記憶體，當 DOS 被存入記憶體時，它大約佔用了 10K 的位置，HIMEM: 就被設定在 DOS 所佔用的位置的下面。圖G－1顯示的就是在不同記憶體大小的情況下，DOS 佔用的記憶體位置。需要注意的是，您必須具有最少 24K 的 RAM 供 DOS 及由磁片或磁帶輸入的 APPLESOFT 使用，而當您想將高解析度的第一頁及 DOS 合用時，您的系統最少需要 32K 的 RAM。圖G－1中同時顯示出由磁碟或磁帶輸入的 APPLESOFT 與高解析度第一頁的互相衝突情形。

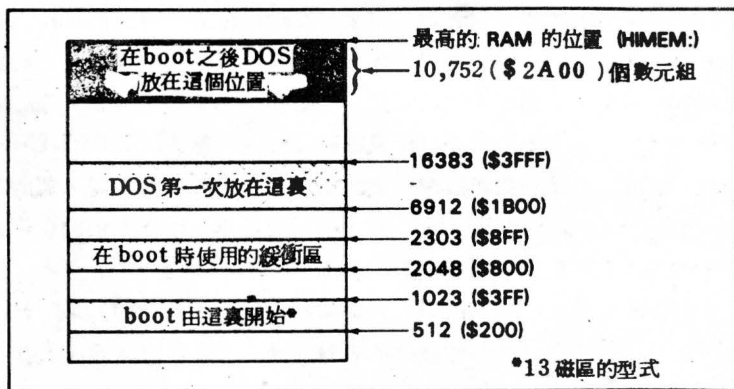
當 DOS 被存入記憶體內時，它使用幾個額外的記憶體區域（參看圖G－2），任何存在這些區域內的資料都會在 DOS 存入後被洗掉。

整數BASIC使用記憶體的情形

整數 BASIC 的程式行由 HIMEM: 的位置開始儲存，就如同圖G－3所顯示的，HIMEM: 的值會自動地因爲您增加、消除或改變程式行而調整。



圖G－1 RAM的使用

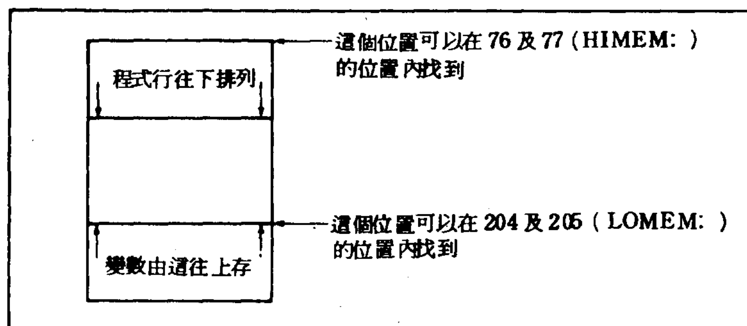


圖G－2 DOS存在記憶體內時記憶體的使用

變數由 LOMEM: 的位置開始存起，當變數的儲存需要改變的時候，LOMEM: 會自動地調整；每一個數值變數都要在儲存體中使用四種指標：變數名稱、DSP 開／關數元組、下一個變數的所在位置，

以及變數的值。

變數名字可以有 100 個字，每一個字在記憶體內使用它的 ASCII 碼代表，而它們的高層次的數元被設定爲 1。



圖G－3　整數BASIC程式的記憶體使用情形

DSP 數元組指出是否 BASIC 命令 DSP 被這個變數使用；這個數元組在正常情況下是 0，當 DSP 被這個變數使用時，這個數元組就被設定爲 1，而當 NO DSP 被執行時，則被設定爲 0。

下一個變數的位置被存在兩個數元組內，低層次的放在前面。

資料按照兩個數元組爲一組的方式儲存，低層次的放在前面；如果這個變數並不是一個矩陣，那麼就只有那麼一對的數元組，如果這個變數是一個矩陣，那麼每一個元素就有一對數元組，依照順序由第 0 個元素開始排列。一個變數及矩陣之間並沒有差別，單一的變數只是矩陣中僅有的第 0 個元素罷了。

字串變數也是依照同樣的方式儲存；變數名稱、DSP 數元組、及下一個變數的位置也依照數值變數的方式儲存。每一個字的 ASCII 碼佔用一個數元組的位置，而它的高層次的數元被設定爲 1，字串的最後一個字的後面是一個表示字串結束的數元組，這個數元組的高層次的數

元被設定爲 0。

Applesoft 使用記憶體

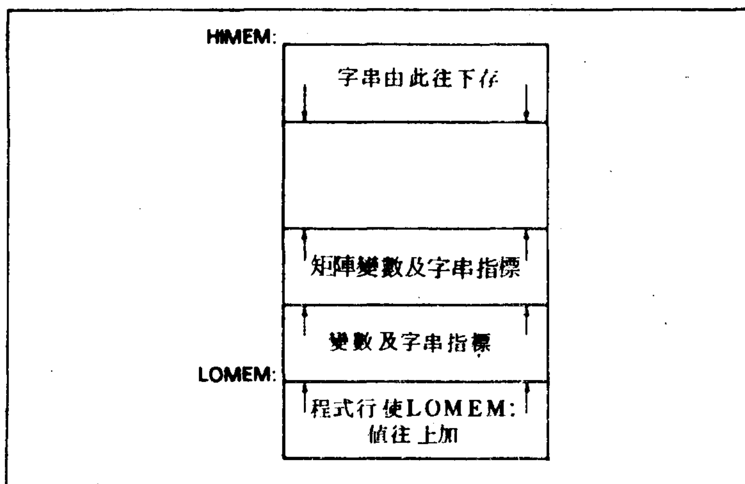
APPLESOFT 的程式行由讀／寫記憶體的結束位置開始放起——由 LOMEM: 設定的值開始，就如圖 G－4 所顯示的一般。當您增加、消除及改變程式行時， LOMEM: 就會自動地調整它的值；簡單的變數及字串的指標都被直接地存在程式行的位置之上，矩陣及字串矩陣的指標則存在變數位置的上面，至於字串的值則由 HIMEM: 的位置開始往下儲存，當您使用了較多的字串值時， HIMEM: 值會自動地往下調整。

每一個數值變數及字串指標使用記憶體七個數元組的位置，每一個實變數使用兩個數元組（ ASCII 碼）代表它的名字（兩個數元組中的高層次的數元都被設定爲 0 ），而它的值則用科學表示儲存，一個數元組用來儲存指數，而利用四個數元組儲存實數部份，它們排列的方式是由最高位到最低位順序儲存數字。

每一個整數變數也使用兩個 ASCII 碼（兩個數元組）代表變數名稱（兩個數元組的高次數元被設定爲 1 ），並且使用兩個數元組儲存這個變數的值——高層次的在前面。

每一個字串指標使用兩個 ASCII 碼（兩個數元組）代表這個變數名字（高次的數元（第一個數元組）被設定爲 1 ，而第二個數元組的高次數元則被設定爲 0 ），使用一個數元組代表字串的長度，兩個數元組儲存字串值的位置，依照低次數元組放在前面的方式排列，至於一個整數變數的最後三個數元組及一個字串指標的最後兩個數元組則不去使用。

數值矩陣及字串指標的矩陣被存在變數的上面，這個變數的名稱是依照 ASCII 碼儲存在前面的兩個數元組內；兩個數元組的高次數元如果都被設定爲 0 則表示實數，若兩者都是 1 則表示整數變數，至於第一個是 1 而第二個是 0 則表示字串指標。



圖G－4　APPLESOFT程式使用記憶體的情形

變數名稱的後面跟著兩個數元組，裏面存著下一個變數的位置，低次的數元組放在前面；接著是儲存這個變數行列數的一個數元組，然後就是表示每一個行列的大小的兩個數元組（高次數元組放在前面），而這個大小則依照相反的方式排列（第一個行列的大小放在最後面）。

然後每一個元素由（0、0、……、0）到（N、N、……、N）依次排列，由最左邊的腳註開始增加。每一個實數的元素使用五個數元組，一個被用來儲存指數，其他的四個用來儲存實數。每一個整數元素則使用兩個數元組，高次數元組放在前面。至於字串指標的元素則使用三個數元組，一個用來代表字串的長度，另外兩個（低次數元組放在前面）則儲存這個字串的位置。

字串的值被放在讀／寫記憶體的最上端，它們每一個字需要佔用一個記憶體的位置，至於重複的字串則只被儲存一次；兩個或更多的指標可以指向同一個字串的位置。當一個新的字串值被設定時，它們就被放

在下一個可以被使用的位置內(HIMEM: 往下調整)。字串即使不再被使用仍然存在記憶體內。您可以使用 FRE 函數強迫一些不被使用的資料聚集起來,而使被丟棄的字串消除並重新設定 HIMEM: 的值。

附錄 H

DISK Ⅱ 的格式

資料被儲存在磁碟上的 35 個同心圓——磁軌（*track*）內，這些磁軌分別用 0 到 34 （ $0 到 $22）的數字代表，每一個磁軌又分爲16個磁區（*sector*），這 16 個磁區分別用 0 到 15 （ $0 到 $F ）的數字代表，每一個磁區可以儲存 256 個數元組的資料。所以在一片磁碟上共有 560 個磁區，總共可以儲存 143,360 個數元組的資料。DOS 爲它本身預留三個磁軌 148個磁區的位置，一個磁軌 116 個磁區儲存磁碟的索引檔，至於剩下的 31 個磁軌 1496 個磁區都被用來儲存資料及程式檔。

DOS 每次傳送資料時都是以一個磁區爲單位的，它在記憶體內使用兩個緩衝區的位置，一個用做讀取資料，一個用做寫出資料。

每一個型式的檔（文字資料、程式及二進位）都有它獨自儲存的格式。文字資料檔是以ASCII 碼的方式儲存的，每一個字佔用一個數元組的位置，然後使用一個值爲 0 的數元組代表這個檔的結束。所有儲存文字資料的數元組在被使用時都以文字資料的型式顯示出來。

一個 BASIC 程式檔的最先兩個數元組內儲存著這個程式的長度——低次數元組放在前面，這個檔的其餘部份則用來儲存這個程式，也是使用ASCII 碼代表。至於一個 APPLESOFT 的檔內，保留字都被代表化（每一個代表數字（*token*）都使用一個ASCII 碼的數元組

），您可以參看附錄 F 有詳盡的對照表。

二進位檔的最先兩個數元組內儲存的是二進位資料在讀／寫記憶體內的開始位置——低次數元組放在前面，下面的兩個數元組內儲存的是這個檔的長度——低次數元組在前，至於其他的部份則用來儲存二進位資料。

磁軌/磁區表

當 DOS 發現一個空白的磁區時，它就會將資料寫入，這也就是說，當一個具有許多磁區的檔被寫入時，它可能被分別儲存在許多分散的磁區內。DOS 會造一個表，表內詳細列出每一檔所使用的磁軌及磁區的位置，而這個表就被稱爲磁軌／磁區表（*track/sector list*）。

在表中的每一個磁區都有一個指標指向下一個磁區的位置，而且每一個磁區都可以指向 122 個被檔所使用的磁區。

一個不再被使用的磁區並不再被存在磁碟內，而在磁軌／磁區表內指向這個磁區的指標內的值被設定爲 0，這使得 DOS 能夠知道不要去尋找這個磁區內的資料。至於磁軌／磁區表內的最後一個磁區內擁有指向最後一個資料磁區的指標，因此，如果第 5000 號記錄是一個隨機處理檔內的第一個，也是僅有的一個記錄，而它的 L 元素是 256（一個磁區存 1 個記錄）。那麼就會有 42 個磁區被使用，第 5000 個記錄內的資料佔用一個磁區的位置，而磁軌／磁區表佔用其餘的 41 個磁區（每一個磁區可以有 122 個指標，那麼 122 乘上 41 得到 5002 個記錄指標），而所有的指標都是 0，只有指向第 5000 個記錄的指標是一個例外。

第 0 個數元組及第 3 到第 12 個的數元組在磁軌／磁區表中都不被使用。第 1 及第 2 個數元組內存有下一個磁區的磁軌及磁區的數字，如果這兩個數元組內存的是 0，那就表示它是表中的最後一個磁區。

索引檔

DOS 使用第 17 個磁軌做爲磁碟的索引檔，對於每一個檔而言，索引檔內存著這個檔的名字、檔的型式、以及這個檔所佔用的磁區數目，這個檔的磁軌／磁區表所在的位置。大多數的這些訊息，您都可以使用**CATALOG**命令在螢光幕上看到。

索引檔的每一個磁區內最多可以存有七個檔的資料。索引檔由第17個磁軌的第 15 個磁區開始，當這個磁區被填滿了之後，繼續往第 14 個磁區儲存下去，一直到第一個磁區爲止，所以索引檔內可以最多擁有 84 個檔的資料。

索引檔的格式都是完全相同的，表H－1將每一個輸入的情形數字化，表H－2則將每一個輸入中的碼所代表的檔的型式做一個說明。

磁軌 17 中的第 0 個磁區並沒有索引檔的輸入，它存的是這個磁碟的辨別資料、硬體的描述及空間的大小。表H－3就是這個磁區的描述。

在磁碟片表（*volume table*）中（表H－3），由第 56 個到 195個的數元組內，每四個數元組內存有一個磁區的空餘位置的資料，當相關的磁區被使用時，那麼代表那個磁區的數元就被設定爲 0，當磁區是空白的時候，那麼這個數元就被設定爲 1。表H－4顯示出那一個數元代表那一個磁區。

表H－1　索引檔的輸入格式

相關的數元組數目	數　元　組　的　內　容
0	這個檔的磁軌／磁區表所在的磁軌數字；當這個檔被消除掉時，這個數元組內就被放入 255。
1	這個檔的磁軌／磁區表所在的磁區數字。
2	檔的型式，參看表H－2。
3 － 32	檔的名稱，使用ASCII碼表示。
33	這個檔所使用的磁區的數目。
34	結束的記號，一般都是 0。

表H－2　索引檔內檔型式的對照

數　元	功　能
0	如果這個數元是 1，則表示這個檔是整數 BASIC 程式檔。
1	如果這個數元是 1，則表示這個檔是APPLESOFT程式檔。
2	如果這個數元是 1，則表示這個檔是二進位檔。
3 － 6	爲以後的擴增預留位置。
7	如果這個數元是 1，則表示這個檔被鎖住了。
如果由第 0 到第 6 個數元都是 0，那麼這個檔就是一個文字資料檔	

表H－3　容積表(第17個磁軌中的第0個磁區)

數元組	描述
0	不被使用。
1	第一個索引檔磁區的磁軌數目。
2	第一個索引檔磁區的磁區數目。
3	DOS 型式的號碼。
4－5	不被使用。
6	磁碟片號碼。
7－38	不被使用。
39	在磁軌／磁區表中一個磁區所能擁有的最大的磁軌／磁區對。
40－47	不被使用。
48－51	可供使用的磁區情形。
52	磁碟中所擁有的磁軌數目。
53	磁碟中所擁有的磁區數目。
54－55	每一個磁區中的數元組的數目：低次數元組在 54，高次數元組在 55。
56－59	第 0 個磁軌中可以被使用的磁區情形。
60－63	第 1 個磁軌中可以被使用的磁區情形。
64－195	由第 2 個到第 195 個磁軌中可以被使用的磁區情形。
196－255	不被使用。

表H－4 可被使用磁區的分布

數元組	數　元	磁　　區
第一個	7	15
	6	14
	5	13
	4	12
	3	11
	2	10
	1	9
	0	8
第二個	7	7
	6	6
	5	5
	4	4
	3	3
	2	2
	1	1
	0	0
第三個	所有的	不 被 使 用
第四個	所有的	不 被 使 用

附錄 I

ASCII碼及 APPLESOFT保留字

在這個附錄中的第一個表就是ASCII 碼及它們所代表的字的對照表，ASCII 碼中的 96 到 127 碼在APPLE Ⅱ 的顯示幕上與 32 到 63 的碼顯示同樣的字，但在其他的輸出設備上則可能顯示出小寫的字。在鍵盤上沒有任何的鍵可以產生 96 到 127 的ASCII 碼。

ASCII 128到 255 碼重複由 0 到 127的碼，在 APPLESOFT 中無法產生這些碼，但是在監督系統及整數 BASIC 中可以產生部份的這些碼。

第二個表列出 APPLESOFT 的保留字。每一個保留字在記憶體中只佔一個數元組的記憶體位置，每一個保留字使用在 128到 255 間的一個碼代替，這個代表的碼在APPLE Ⅱ 的記憶體及磁碟內代替保留字。這個表是按照代表碼的數字順序排列的。在附錄 F 中則是依照保留字的字母排列順序排列的。

ASCII字碼

ASCII碼	顯示字	鍵	ASCII碼	顯示字	鍵
0		Ctrl-@	48	0	0
1		Ctrl-A	49	1	1
2		Ctrl-B	50	2	2
3		Ctrl-C	51	3	3
4		Ctrl-D	52	4	4
5		Ctrl-E	53	5	5
6		Ctrl-F	54	6	6
7	(bell)	Ctrl-G	55	7	7
8	(backspace)	Ctrl-H or ←	56	8	8
9		Ctrl-I	57	9	9
10	(linefeed)	Ctrl-J	58	:	:
11		Ctrl-K	59	;	;
12		Ctrl-L	60	<	<
13	(carriage return)	Ctrl-M	61	=	=
14		Ctrl-N	62	>	>
15		Ctrl-O	63	?	?
16		Ctrl-P	64	@	@
17		Ctrl-Q	65	A	A
18		Ctrl-R	66	B	B
19		Ctrl-S	67	C	C
20		Ctrl-T	68	D	D
21	(forward space)	Ctrl-U or →	69	E	E
22		Ctrl-V	70	F	F
23		Ctrl-W	71	G	G
24	(cancel line)	Ctrl-X	72	H	H
25		Ctrl-Y	73	I	I
26		Ctrl-Z	74	J	J
27		Esc	75	K	K
28		n.a.	76	L	L
29		Ctrl-Shift-M	77	M	M
30		Ctrl-^	78	N	N
31		n.a.	79	O	O
32	space	space bar	80	P	P
33	!	!	81	Q	Q
34	"	"	82	R	R
35	#	#	83	S	S
36	$	$	84	T	T
37	%	%	85	U	U
38	&	&	86	V	V
39	'	'	87	W	W
40	(	(	88	X	X
41	)	)	89	Y	Y
42	*	*	90	Z	Z
43	+	+	91	[	n.a.
44	,	,	92	\	n.a.
45	-	-	93	]	Shift-M
46	.	.	94	^	^
47	/	/	95		n.a.

注意：如果可以由監督系統或整數BASIC產生，則加入128到ASCII碼內。 n.a. = 不能由APPLE Ⅱ鍵盤產生。

APPLESOFT 保留字及代表數字

代表數字	保留字	代表數字	保留字	代表數字	保留字
128	END	164	LOMEM:	200	+
129	FOR	165	ONERR	201	-
130	NEXT	166	RESUME	202	*
131	DATA	167	RECALL	203	/
132	INPUT	168	STORE	204	^
133	DEL	169	SPEED=	205	AND
134	DIM	170	LET	206	OR
135	READ	171	GOTO	207	>
136	GR	172	RUN	208	=
137	TEXT	173	IF	209	<
138	PR#	174	RESTORE	210	SGN
139	IN#	175	&	211	INT
140	CALL	176	GOSUB	212	ABS
141	PLOT	177	RETURN	213	USR
142	HLIN	178	REM	214	FRE
143	VLIN	179	STOP	215	SCRN(
144	HGR2	180	ON	216	PDL
145	HGR	181	WAIT	217	POS
146	HCOLOR=	182	LOAD	218	SQR
147	HPLOT	183	SAVE	219	RND
148	DRAW	184	DEF	220	LOG
149	XDRAW	185	POKE	221	EXP
150	HTAB	186	PRINT	222	COS
151	HOME	187	CONT	223	SIN
152	ROT=	188	LIST	224	TAN
153	SCALE=	189	CLEAR	225	ATN
154	SHLOAD	190	GET	226	PEEK
155	TRACE	191	NEW	227	LEN
156	NOTRACE	192	TAB(	228	STR$
157	NORMAL	193	TO	229	VAL
158	INVERSE	194	FN	230	ASC
159	FLASH	195	SPC(	231	CHR$
160	COLOR=	196	THEN	232	LEFT$
161	POP	197	AT	233	RIGHT$
162	VTAB	198	NOT	234	MID$
163	HIMEM:	199	STEP		

附錄 J

換 算 表

這個附錄包含下面的換算表：

- 十六進位——二進位數字
- 十六進位——十進位數字

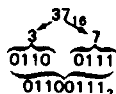
十六進位—二進位換算表

您可以使用下面的表換算 0 — 0F 的十六進位數字與 0000—1111 的二進位數字。

將較大的二進位數字轉換成十六進位數字的方法就是每次以四個二進位數字為單位，由右自左做換算的工作，如果在左邊的區內不足四位數字，則在前面加上零補足位數。下面就是一個例子：

$$100101_2 = \underbrace{\underbrace{0010}_{2}\underbrace{0101}_{5}}_{25_{16}}{}_2$$

將十六進位數字轉換成二進位數字的方法則是每次轉換一個數字。下面是一個例子：



十六進位	二進位
00	0000
01	0001
02	0010
03	0011
04	0100
05	0101
06	0110
07	0111
08	1000
09	1001
0A	1010
0B	1011
0C	1100
0D	1101
0E	1110
0F	1111

下面這個表可以直接由十六進位的數字轉換成十進位的數字，它的範圍是十六進位的0到 FFF ，轉換成十進位的0 － 4095 。至於較大數字的轉換，您可以增加接著的這個表。

十六進位—十進位整數換算表

十六進位	二進位	十六進位	二進位
01 000	4 096	20 000	131 072
02 000	8 192	30 000	196 608
03 000	12 288	40 000	262 144
04 000	16 384	50 000	327 680
05 000	20 480	60 000	393 216
06 000	24 576	70 000	458 752
07 000	28 672	80 000	524 288
08 000	32 768	90 000	589 824
09 000	36 864	A0 000	655 360
0A 000	40 960	B0 000	720 896
0B 000	45 056	C0 000	786 432
0C 000	49 152	D0 000	851 968
0D 000	53 248	E0 000	917 504
0E 000	57 344	F0 000	983 040
0F 000	61 440	100 000	1 048 576
10 000	65 536	200 000	2 097 152
11 000	69 632	300 000	3 145 728
12 000	73 728	400 000	4 194 304
13 000	77 824	500 000	5 242 880
14 000	81 920	600 000	6 291 456
15 000	86 016	700 000	7 340 032
16 000	90 112	800 000	8 388 608
17 000	94 208	900 000	9 437 184
18 000	98 304	A00 000	10 485 760
19 000	102 400	B00 000	11 534 336
1A 000	106 496	C00 000	12 582 912
1B 000	110 592	D00 000	13 631 488
1C 000	114 688	E00 000	14 680 064
1D 000	118 784	F00 000	15 728 640
1E 000	122 880	1 000 000	16 777 216
1F 000	126 976	2 000 000	33 554 432

至於十六進位的小數部份，您可以依照下面的規則做轉換的工作：

1. 將十六進位的小數部份變成整數乘上 16^{-n} 的型式， n 就是這個小數所佔的位數。

$$0.CA9BF3_{16} = CA9BF3_{16} \times 16^{-6}$$

2. 將這個十六進位整數轉換成十進位的整數。

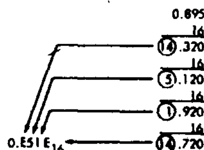
$$CA9BF3_{16} = 13\,278\,195_{10}$$

3. 將這個十進位的數值用 16^{-n} 值去乘。

$$\begin{array}{r} 13\,278\,195 \\ \times\, 596\,046\,448 \times 10^{-16} \\ \hline 0.791\,442\,096_{10} \end{array}$$

十進位的小數部份也可以轉換成十六進位的小數，您可以連續地使用 16_{10} 去乘前面所得的乘積，在每一次相乘之後，整數的部份就被移作十六進位的小數部份了。需要注意的是，得到的整數部份在被移往當做小數之前，必須先轉換成十六進位的數字型式。

例子：將 0.895_{10} 轉換成十六進位小數



十六進位—十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

十六進位—十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

十六進位一十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

十六進位一十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

十六進位一十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2958	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	[illegible]	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

十六進位—十進位整數換算表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

附錄 K

參考書籍

BASIC語言

Albrecht, Finkle, and Brown. *BASIC*. Peoples Computer Company, Menlo Park, California, 1967.

Coan, James S. *Advanced BASIC*. Hayden Book Co., Rochelle Park, New Jersey.

Coan, James S. *Basic BASIC*. Hayden Book Company, Rochelle Park, New Jersey.

Dwyer, T. *A Guided Tour of Computer Programming in BASIC*. Houghton Mifflin Company, 1973.

Kemeny, J., and Kurtz, T. *BASIC Programming*. Peoples Computer Company, Menlo Park, California, 1967.

Pegels, C. *BASIC: A Computer Programming Language*. Holden-Day, Inc., 1973.

Peoples Computer Company. *What to Do After You Hit Return*. Menlo Park, California.

Sack, J., and Meadows, J. *Entering BASIC*. Science Research Associates, 1973.

組合語言程式設計

Leventhal, Lance A. *6502 Assembly Language Programming*. Osborne/McGraw-Hill, Berkeley, California, 1979.

Osborne, A. *An Introduction to Microcomputers: Volume 1 — Basic Concepts.* 2nd ed., Osborne/McGraw-Hill, Berkeley, California, 1980.

Scanlon, Leo J. *6502 Software Design.* Howard W. Sams, Indianapolis, Indiana.

Zaks, Rodnay. *6502 Applications Book.* Sybex, Berkeley, California.

定期出版的雜誌

Apple Orchard, P. O. Box 1493, Beaverton, Oregon 97075.

"Apple-Cart," *Creative Computing.* P.O. Box 789-M, Morristown, New Jersey 07960.

CALL A. P. P. L. E., 304 Main Ave. S., Suite 300, Renton, Washington 98055.

Compute!, P.O. Box 5406, Greensboro, North Carolina 27403.

Micro, P.O. Box 6502, Chelmsford, Massachusetts 01824.

Nibble. P.O. Box 325, Lincoln, Massachusetts 01773.

Personal Computing, P.O. Box 13916, Philadelphia, Pennsylvania 19101.

Purser's Magazine. P.O. Box 466, El Dorado, California 95623.

Softalk. 10432 Burbank Bl., N. Hollywood, California 91601.

APPLE的書刊

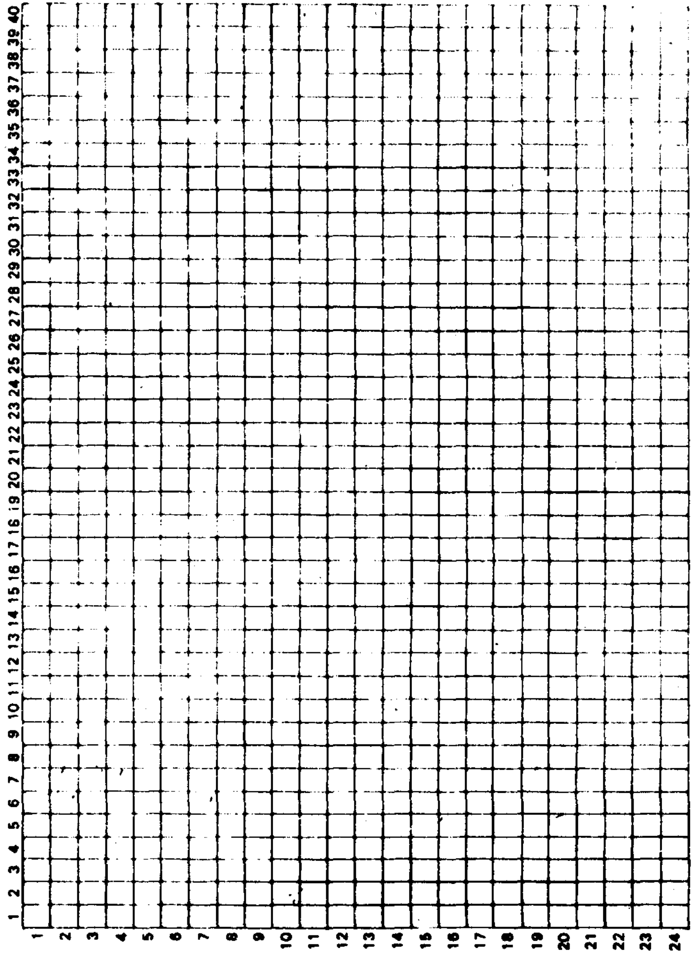
Apple II/II Plus Reference Manual	A2L0001A
Parallel Printer Interface Manual	A2L0004
Apple II BASIC Programming Manual	A2L0005
Applesoft II Reference Manual	A2L0006
Communications Interface Manual	A2L0007
High-Speed Serial Interface Manual	A2L0008
Disk II Floppy Disk Manual (DOS 3.2.1)	A2L0012
Applesoft Tutorial Manual	A2L0018
Graphics Tablet Manual	A2L0033
Silentype Manual	A2L0034
The DOS Manual (DOS 3.3)	A2L0036

附錄 L

螢光幕顯示的格式

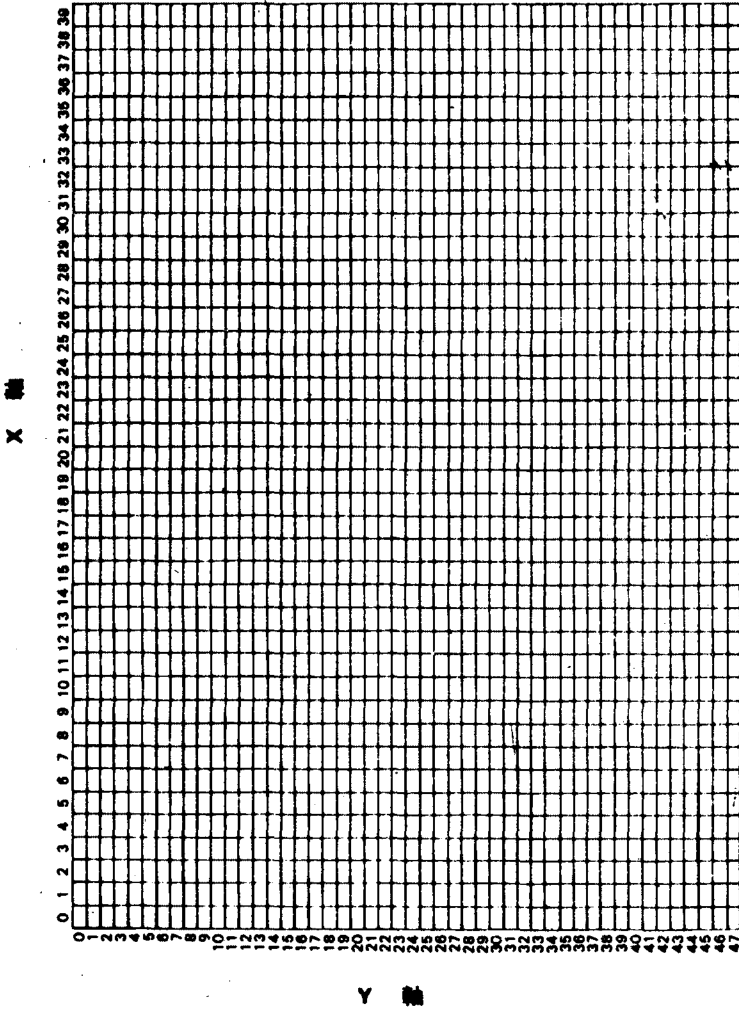
您可以使用這個附錄中的格式設計螢光幕顯示的情形。在文字資料的顯示情況下，行及列的起始號碼都是由 1 開始，這使得文字資料的顯示較為方便。至於在低解析度圖形的顯示情況時，行及列的開始位置都是由 0 開始，這是爲了適應低解析度圖形的命令而定的。

水平位置



垂直位置

文字資料顯示幕


X 軸
Y 軸
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

低解析度圖形顯示幕

附表　港、台与内地若干计算机名词术语对照

港　台	内　地	英　译
2～4　划		
及	"与"	AND
二进制系统	二进制记数制	binary system
二进编十进码	二-十进制、二进制编码的十进制	BCD
及闸	"与"门	AND gate
下压迭	下推栈	push-down stack
小型积体电路	小规模集成电路	SSI
大型积体电路	大规模集成电路	LSI
分工	多路化，分成多路	demultiplexing
介面	接口	interface
文数码	字母数字代码	alphanumeric code
反及闸	"与非"门	NAND gate
反或闸	"或非"门	NOR gate
互斥或	"异"	XOR
分离元件	分立元件	discrete component
闩持电路	锁存电路	latch circuit
仃顿认可	保存肯定	hold acknowledge
巴士界面	总线接口	bus interface
双排包装	双列直插式组件	dual in-line package
中型积体电路	中规模集成电路	MSI
双向汇流排	双向总线	bidirectional bus
双极系的I^2L	集成注入逻辑	integrated injection logic
不在意情况	无关条件，自由选取条件	don't care condition
中央处理单位（中央处理系统）	中央处理机	CPU
5～6　划		
电闸	门	gate
电驿	继电器	relay
长度	字长	word length
主奴	主从的	master-slave
汇流道（巴士）	总线	bus

主程式	主程序	main program
目的码	结果代码	object code
主档案	主文件	master file
卡氏图	卡诺图	Karnaugh map
记录器	寄存器	register
正反器	触发器	flip - flop
记忆图型	存储变换	memory map
布林变数	逻辑变量，布尔变数	Boolean variable
发光二极体	发光二极管	LED
可换性（匹配）	兼容性，相容性	compatibility
可程式唯读记忆体	可编程序的只读存储器	PROM
可清洗以重定内容的PROM	可擦可编程序只读存储器	EPROM
可经由电改变全部或部份内容的ROM	可改写的只读存储器	EAROM
可规划的晶体组合	可编程序逻辑阵列	PLA
可制程式通信用介面	可编程序的通信接口	Programmable communication interface
可制程式周边介面	可编程序外围接口	Programmable Peripheral interface
电晶体－电晶体逻辑	晶体管晶体管逻辑	TTL
伪	假（ALGOL中的布尔值）	false
字元	字符，符号	character
存入	存储	store
多工	多路转换，多路化	multiplexing
多工器	多路转换器	multiplexer
次常式(副程式,常式)	子程序	subroutine
执行消除	删去，删除	delete
杂音限度	噪音容限	noise margin
延迟电路	时延电路	time delay circuit
同位数元	奇偶检验位	parity bit
许可岔断	允许中断	EI
地(位)址汇流道	地址总线	address bus
自己程式计划	自动编程序	self programming
动态读写记忆体	动态随机存取存储器	dynamic RAM

7～8 划划

位址	地址	address
位元	位（二进制数的）	bit
串接	串行的，串联的	serial
岔断	中断	interrupt
抛投	转态过程	polling
位元组	位组	byte
运算码	操作码	OP code
运算物	操作数，运算对象	operand

初期化	初始化	initialize
芬氏图	文氏图	venn diagram
状广旗标	状态特征位,状态标志	status flag
串接资料	串行数据	serial data
针冲脉波	一闪信号,假信号	glitch
岔断认可	中断应答	interrupt acknowledge
乱数产生器	随机发生器	random number generator
位元分割型	位片型	bit-slice stencil
时计产生器	时钟脉冲发生器	clock generator
狄莫根定理	德·摩尔根定理	De Morgn's theorem
更大型积体电路	超大规模集成电路	VLSI
系统资料汇流带	系统数据总线	system-data bus
软体	软件	software
迭器（堆层）	堆栈，存储栈	stack
非同步	异步	asynchronous
易记码	助记码	mnemonic code
受讯者	收听者	listener
奇价位	奇数奇偶校验	odd parity check
终止数元	停止位	stop bit
周边装置	外围设备,外部设备	peripheral equipment; peripheral device
组合语言	汇编语言	assembly language
定时频率	时钟脉冲频率	clock frequency
组合程式	汇编程序	assembler
实用常式	实用程序	utility routine
直接记忆出入	直接存储器存取	DMA
组合语言程式	汇编语言程序	assembly language program
实时间监视程式	实时监督程序	real time monitor
单向汇流带	单向总线	one way bus
迭器指标记录器	栈指示字	stack pointer

9～11 划

重置，复始	复位，置“○”	reset
除错	排除错误(程序的)	debug
脉波	脉冲	pulse
标名	标号	label
指令集	指令系统	instruction set
类比	模拟	analog
査知器(感知器)	读出器，传感器	sensor
指标记录器	变址寄存器	index register
指令不许可岔	禁止中断	DI
资料	数据	data
倒转	反绕(磁带等的)	rewind
竞跑	竞态	race
核对数元	检验位	check bit

积体电路	集成电路	I C
高阶语言	高级语言	high-level language
脉波注入器	逻辑脉冲发生器	logic pulser
资料汇流道	数据总线	data bus
资料暂存器	数据寄存器	data register
起始数元	起动位	start bit
接脚	管脚	pin
掩罩	屏蔽	mask
累积器	累加器	accumulator
偶价位	偶数奇偶校验	even parity check
假指令	伪指令	pseudo instruction
控制单位	控制器	control unit
符号位元	符号位	sign bit
副常式召唤	子程序调用	subroutine call
控制汇流道	控制总线	control bus
移位暂存器	移位寄存器	shift register
唯读记忆体	只读存储器	read only memory
第二级记忆	辅助存储器	secondary storage auxiliary storage

12 划以上

硬体	硬件	hardware
程式	程序	program
暂停点	断点	break point
晶片选择	芯片选择,选片	chip select
编辑程式	编辑程序	editor
智慧终端机	智能终端设备	intelligent terminal
程式置定器(程式设计器)	可编程序只读存储器的程序编制器	PROM programmer
握手式输入输出	信号交换输入输出	handshaking I/O
福传	FORTRAN语言,公式翻译程序语言	FORTRAN
解码	译码	decode
简字符号	助记符号	mnemonic symbol
解多工器	多路分配器,译码器	demultiplexer
罩幕形成的ROM	掩模可编程序的只读存储器	mask programmable ROM
旗号正反器	标记触发器	flag flip-flop
算术逻辑单位	运算器,算术及逻辑运算部件	ALU
磁碟	磁盘	disc, disk
激发表	激励表	excitation table
随意逻辑	随机逻辑	random logic
静态读写记忆体	静态随机存取存储器	static RAM
翻新(刷新)	更新,再生	refresh
覆用常式	重入子程序	reentrant subroutine

内部交流

G 19/5 APPLE Ⅱ使用手册

(中3—5/69)

D00300